

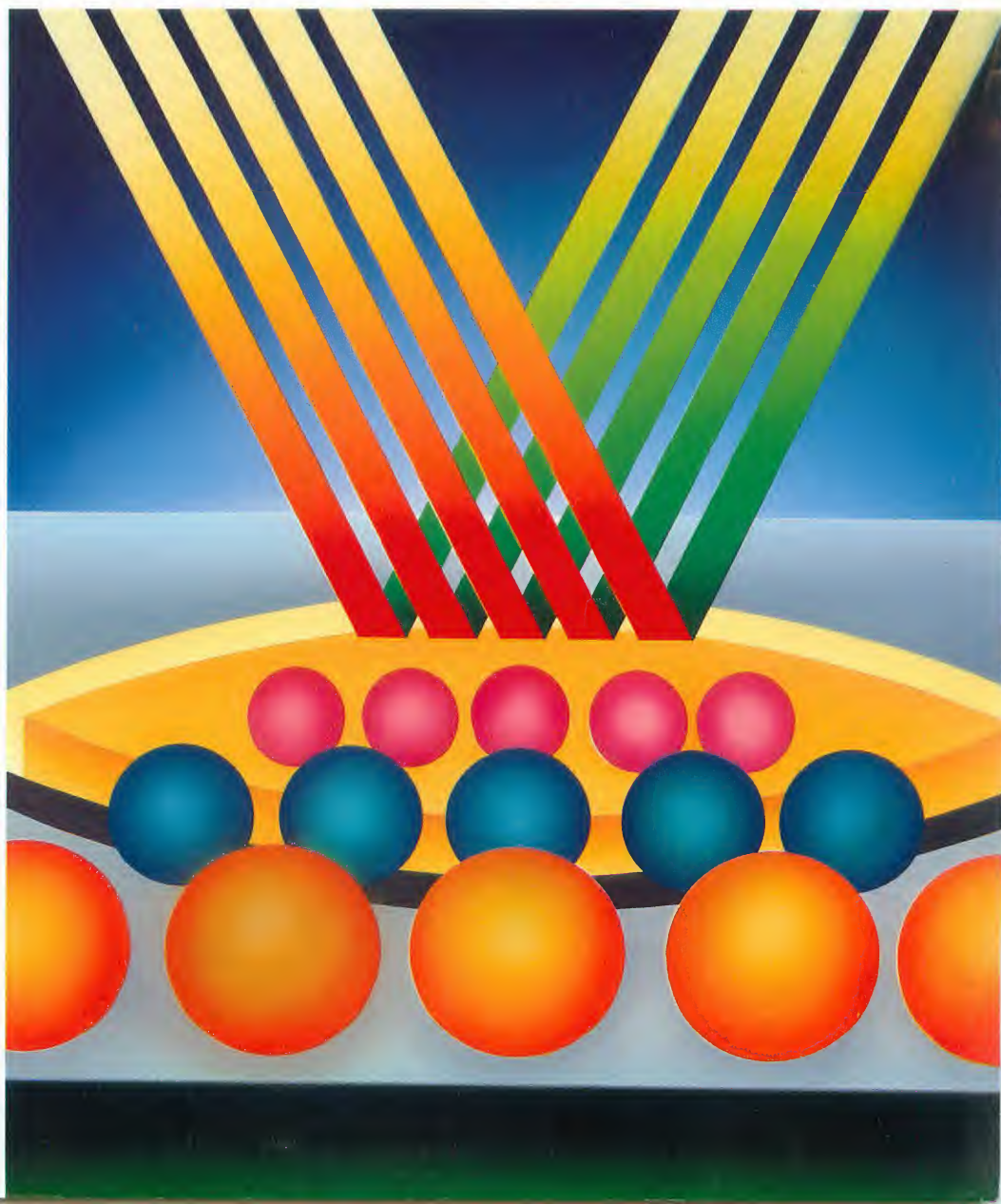
# トランジスタ技術

SPECIAL

No.27

## 特集 ハードディスクとSCSI 活用技術のすべて

本格活用のためのハード&ソフトのすべてを詳解



好評発売中

# トランジスタ技術 SPECIAL No.26

## 特集 68000ソフト & ハードのすべて

実用ライブラリの作成と便利チップ68301/68303の活用技術

B 5 判 160頁 定価1,540円 送料260円



MC68000は、当初ワーク・ステーションなどの分野で使用されてきましたが、最近では組み込み用などの制御用ボードに搭載されることも多くなっています。これも68000マイクロプロセッサの理解しやすく拡張性のあるコンピュータ・アーキテクチャによるものでしょう。

本書は68000のハードウェア、ソフトウェアの参考書として役立つことを目的としています。特集では、68000のハードウェアの基本的な使い方から、68000の新たな方向性を示す68301の解説と68303の全容を紹介し、また、68000アセンブラ・プログラミングの基礎としてC言語の主なライブラリ関数をアセンブラで記述することやデバッグ・モニタの作成をとりあげます。さらに32ビット・マイクロプロセッサ68020を使った簡単なボードを製作し、68000と68020のアーキテクチャの相違点について解説します。

既刊 (B 5 判 2 色刷 定価1,540円 送料260円)

- |                               |                            |
|-------------------------------|----------------------------|
| No 1 個別半導体素子活用法のすべて           | No 14 技術者のためのCプログラミング入門    |
| No 2 作りながら学ぶMC68000           | No 15 アナログ回路技術の基礎と応用       |
| No 3 PC9801と拡張インターフェースのすべて    | No 16 A-D・D-A変換回路技術のすべて    |
| No 4 C-MOS標準ロジックIC活用マニュアル     | No 17 OPアンプによる回路設計入門       |
| No 5 画像処理回路技術のすべて             | No 18 ホビー・エレクトロニクス入門       |
| No 6 Z80ソフト&ハードのすべて           | No 19 PC9801計測インターフェースのすべて |
| No 7 HD64180徹底活用マニュアル         | No 20 アナログ回路シミュレータ活用術      |
| No 8 データ通信技術のすべて              | No 21 デジタル・オーディオ技術の基礎と応用   |
| No 9 パソコン周辺機器インターフェース詳解       | No 22 デジタル回路ノイズ対策技術のすべて    |
| No 10 IBM PC & 80286のすべて      | No 23 回路デザイナーのためのPLD最新設計法  |
| No 11 フロッピー・ディスク・インターフェースのすべて | No 24 Cによる組み込み機器用プログラミング   |
| No 12 入門ハードウェア 手作り測定器のすすめ     | No 25 最新マイコン・メモリ・システム設計法   |
| No 13 シミュレータによる電子回路理論入門       |                            |

年間購読をご希望の方は、年間購読料10,500円(送料、税込)を現金書留または郵便振替(東京0-10665)で、CQ出版(株)経理部までお申し込みください。バックナンバーは最寄の書店から注文できます。弊社へ直接ご注文の場合は、定価、送料を添えてCQ出版(株)営業部宛へお願いいたします。

CQ出版社

〒170 東京都豊島区巣鴨1-14-2

出版部 ☎(03)5395-2121  
営業部 ☎(03)5395-2141

振替 東京0-10665

(定価は税込です)



# トランジスタ技術

CONTENTS

## SPECIAL No.27

### 特集 ハードディスクと SCSI 活用技術のすべて

本格活用のためのハード&ソフトのすべてを詳解

第0章	ハードディスクの歩み ●松村清明	2
	容量、スピード、価格、体積、性能への挑戦	

第1章	ハードディスクの機構部	
	箱の中身はこれほどまでに複雑なのだ	
	●杉本行正/中村正夫/望月将伸/宮崎 修/小寺 勉/渡辺敬明	6

第2章	ハードディスク上への信号記録/再生 ●松村清明	12
	記録/再生もカセットみたいに単純ではない	

APPENDIX	代表的なHDC	17
----------	---------	----

第3章	記録/再生信号の変調と復調 ●松村清明	19
	マイクロンセンサと訂正符号は信号記録技術の基本	

第4章	ハードディスクの高速インターフェース ●松村清明	26
	SASI, SCSIが今のところ標準	

APPENDIX	代表的なSCSIコントローラ	53
----------	----------------	----

第5章	SCSIを使ってハードディスクをつなぐ ●松村清明	55
	ハードディスクを使うための作業はこれだけ	

第6章	自分で作るSCSIのハード&ソフト	
	ハードディスク&SCSIはアマチュアでも実現可能	
	●宇野沢成夫/橋家鶴蔵/鈴木 洋	85

APPENDIX	SPC(MB89352)の解説	136
----------	-----------------	-----

第7章	SCSI2の概要—拡張と変更を中心に ●菅谷誠一	140
	もう一部ではSCSI3の話も、乗り遅れるな!	

編集雑記		164
------	--	-----

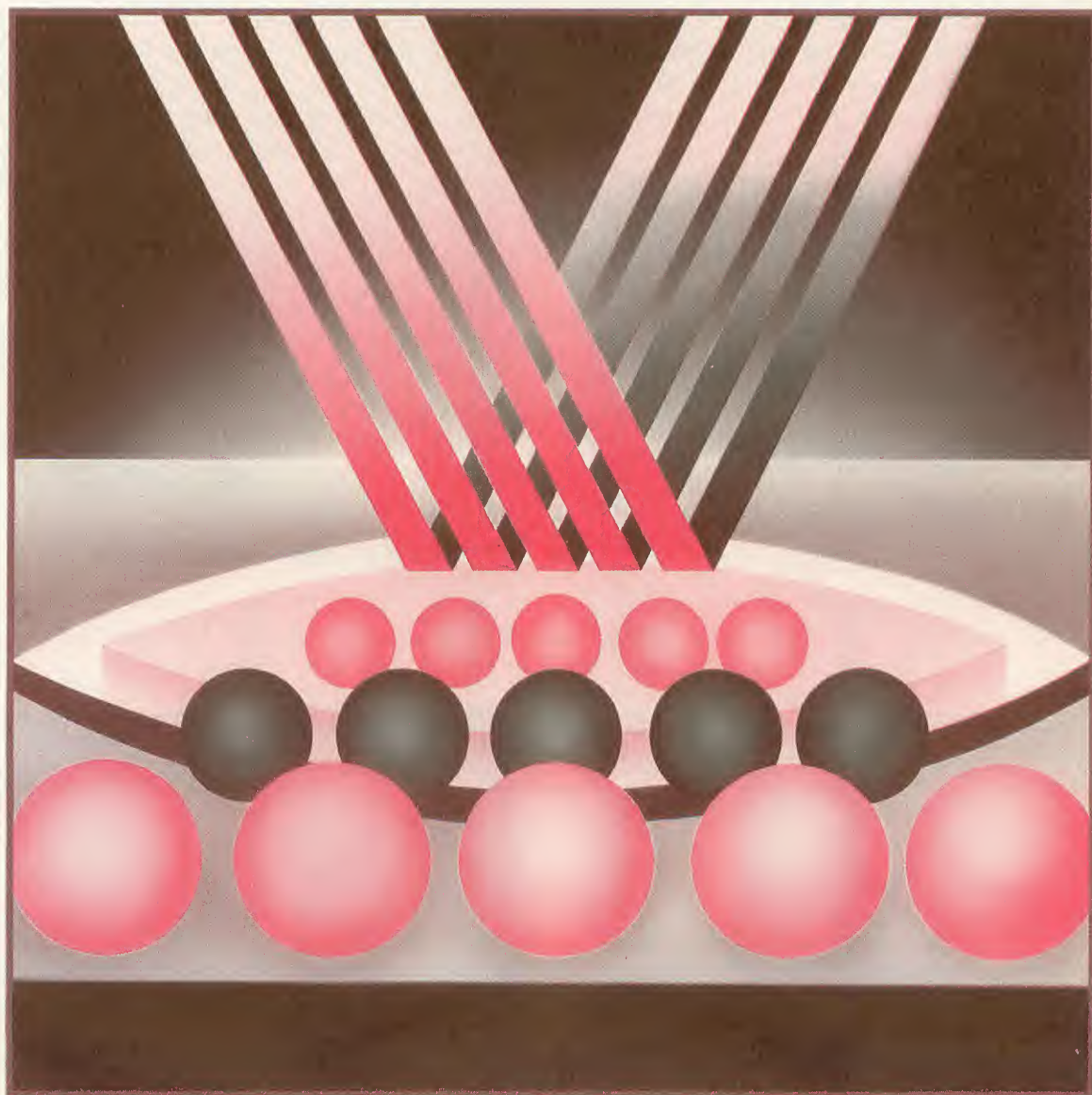




# 特集 ハードディスクとSCSI 活用技術のすべて

■ 本格活用のためのハード&ソフトのすべてを詳解 ■

ハードディスクはいつの間にかパソコンの必需品になりました。パーソナル仕様 HDD の密度と速度はとどまることなく向上し、値段と体積がみるみるうちに下降しています。そんな現在、ハードディスク関連の技術資料は皆無に等しく、内部についても見てくれと同じくブラックボックス化しています。今回の特集では全く中身について知られていない HDD のしくみと HDD とは切り離せない関係にある SCSI をあわせて解説します。



## 第0章

# ハードディスクの歩み

容量、スピード、価格、体積、性能への挑戦

松村清明

1975年頃、14インチのハードディスクが付いたミニコンピュータが5千万円くらいしました。

このハードディスクの容量は5Mバイトだったと記憶しています。入出力端末はテレタイプでした。

5Mバイトでは大きなファイル・システムをのせることができません。そのため限られた実行形式ファイルとコンパイルのためのテンポラリー・ファイルのみがディスク上に置かれ、ソース・プログラムは紙テープで保存していました。

パソコンに600Mバイトのハードディスクが接続され、40Mバイトのハードディスクがディスカウント・ショップで山と積まれて売られている現在とは隔世の感があります。

もちろん、このような変化が急に起こったわけではありません。パソコンの発展とともにハードディスクも多くの小さなステップの積み重ねで発展してきたわけです。

### ●パソコン用ハードディスク

8インチ250Kバイトのフロッピー・ディスクとインテルのCPU8080AがCP/Mのもとでディスク・オペレーティング・システムとして動作したとき、システムとしての能力はそれ以前の紙テープ・ベースのミニコンピュータ・システムの能力を超えてしまいました。

これは8080AとCP/Mという共通プラットフォームの上に積み上げられたソフトウェアの重みと同時に、ディスク・オペレーティング・システムを簡単に構築することを可能にした8インチ・フロッピー・ディスク・ドライブによるところが大きい。

1980年、ICMが8インチ20Mバイトのハードディスク・ドライブを製品化し、NECのパソコンPC8001に接続して100万円位の価格で販売開始しました(写真1)。これはパソコン用のハードディスクとしては日本で初めてのものでした。フロッピー・ディスクと比べるとかなり高速でしたが、ホストCPU(Z80A)の能力が低いので、アプリケーション上で使用しても10倍の高速化というわけにはいきませんでした。

しかし、当時すでに発売されていた8086にはソフトウェアがなく、ホストCPUと周辺機器のアンバランスが目立った時代でした。

この製品はミニコンピュータ用のディスクと比べると圧倒的に安かったのですが、CPUの能力も合わせたトータルなスループットが低く、一般的に使用されるまでにはなりませんでした。

### ●PC9801用ハードディスク

1982年になるとST412タイプの5.25インチ10Mバイト・ハードディスクが日本でも入手可能になりま



〈写真1〉初期のパソコン用ハードディスク



〈写真2〉1トラック分のバッファを内蔵したハードディスク



した。ドライブ上には8ビット・シングルチップCPUが搭載されており、これがドライブのシークをコントロールしていました。このドライブにSASIインターフェースのディスク・コントローラを接続し、SASIバスでパソコンと接続しました。価格が50万円を切ったのもこの頃です。

また初代PC9801が発売され、ハードディスクを接続するパソコンも急速にPC8801からPC9801へと移行していきました。パソコン用のハードディスクでは先行していたサードパーティの物に加えて、NECもPC9801用ハードディスクを発売しました。

また、新機種PC9801EやPC9801Fではハードディスクからもブート・アップできるようにBIOSが変更されたことで、PC9801は名実ともにハードディスク対応のパソコンとなりました。

### ●ハードディスクの高速化

1984年頃にはPC9801のスピードも高速化され、周辺機器の高速化がパソコンのトータル・スループットを考えた上で重要になってきました。

そこで、インターリーブを使って5～6回転で1トラックのデータをアクセスしていたそれまでの方法に対し、1トラック分のバッファをハードディスク・コントローラに持たせることで、高速リード/ライトを可能にしたハードディスク・ユニットも市場に登場しました(写真2)。

1986年頃になるとパソコンのメイン・メモリも640Kバイト・フル実装が一般化し、数100Kバイトのプログラム・サイズが一般化しました。そこで、プログラム起動時間がパソコンの操作感を左右するものとして重要になってきました。

プログラム起動時間、すなわちプログラム・ロードのスピードを決定するのはディスクのシーケンシャル・アクセス・スピードです。そこで、2トラック分のバッファを用意し、これをキャッシュ・バッファとし

て、ホスト側のデータ転送レートに影響されずにつぎのトラック分のデータをリード、もしくは前のトラックをライトできるようにして高速シーケンシャル・リード/ライトを可能にするといった方法も使われました(写真3)。

### ●普及型ハードディスク

そのうちハードディスクの価格はどんどん低下し、1987年には実勢価格で10万円以下のハードディスクが登場しました(写真4)。

この頃になるとこの価格のディスクを作るためにコントローラとディスク・ドライブを一体化し、性能的にはそれまでのハードディスクと同じものを3.5インチ・ディスクのサイズに押し込んでいました。

この頃のハードディスク・コントローラはすでにASICの塊になっていましたが、さらに複数のASICを複合化し、コンパクト化しました。

このように開発に多くの労力を費やしたものの、価格的なインパクトは大きく、市場規模は予想以上に拡大しました。

### ●操作環境

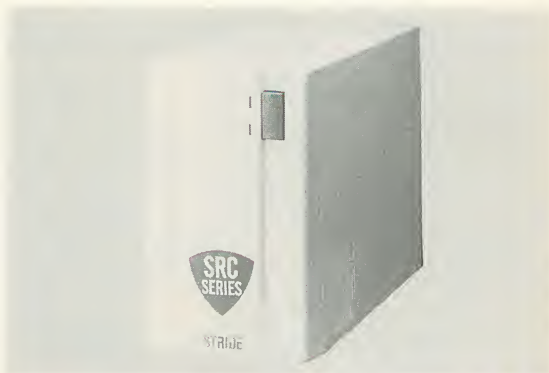
市場規模の拡大と共にハードディスクのユーザもパソコンをコンピュータとして使用する人達から、ブックボックス化されたターン・キー・システムとして使用する人達へ広がってきました。

こういった人達にとってすればハードディスクもソフトウェアを使うための単なる手段です。したがってソフトウェアのインストールさえできればとりあえずハードディスクの使い方を覚える必要はありません。

そこで、画面に出てくる指示にしたがって操作を行えば、目的のソフトウェアがインストールされ、そのソフトウェアがハードディスク上で使用可能になるような操作環境(写真5)があたりまえのこととして各社から提供されています。



〈写真3〉キャッシュ・バッファの内蔵ハードディスク



〈写真4〉普及型ハードディスク

この頃から、パソコン用ハードディスクというもののとらえ方がハードディスクという単なるハードウェアを提供するのではなく、ソフトウェアを快適に使用するための環境を提供するのだという方向に変わってきたわけです。

## ●インターフェースの高速化

コンピュータの世界では高速化の要求はとどまるどころを知りません。

1987年には1M~16Mバイトのディスク・キャッシュを持ち、16ビットCPUであるV40でコントロールするハードディスク・コントローラを開発し、製品化しました。

しかし、ディスクをここまで高速化すると今まで表に現れていなかったことが問題になりました。PC9801のハードディスク・インターフェースのデータ転送レートがDMA転送速度のボトル・ネックで限界に達してしまったのでハードディスクをいくら高速化してもほとんどスピード・アップしないのです。

そこで、ディスク・インターフェースのデータ転送レートを上げる方法を考えないとせっかく作った製品も出荷できないということになりました。

こうともなると、さて困ったものです。PC9801ではDMAコントローラ(i8237)の関係で、シングル・トランスファ・モードDMAのみしか使えません。これではデータ・レートは最大速度で400Kバイト/sくらいのもので、これに対してキャッシュ・ヒットによるメリットを出すためにはせめて800Kバイト/sくらいの転送速度が必要です。そこで、Vシリーズや286以上のCPUで可能な入出力ストリング命令を用いてCPUにデータ転送させることにしました。この方法は少しずつ転送しても効果がありません。256バイト以上のような単位でステータス・チェックなしに転送できることが必要です。そこで、ゲートアレイとSRAMを用いてFIFOを構成し、このSRAMの中

にバッファリングする転送方法を実現しました。

1988年には記録方式をMFM変調方式からRLL(2,7)変調方式へ変更すると同時に、データ転送レートを上げた物へと移行しました。これにより、5Mbpsのデータ転送レートが10Mbpsになったことで同一ヘッド、シリンダ数で2倍の記録容量となり、20万円以下で80Mバイトというディスクを実現しました。

そして1990年にいたると2.5インチ40Mバイト・ハードディスクが登場します(写真6)。

## ●ハードディスクの将来

さて、ハードディスクはこれからどのようなのでしょうか。他のメモリに置き換えられてしまうのでしょうか。

今までも幾度となく繰り返されてきた疑問ですが、ハードディスクはいまだに、同等もしくはより大きくなって二次記憶装置の主流をなしています。

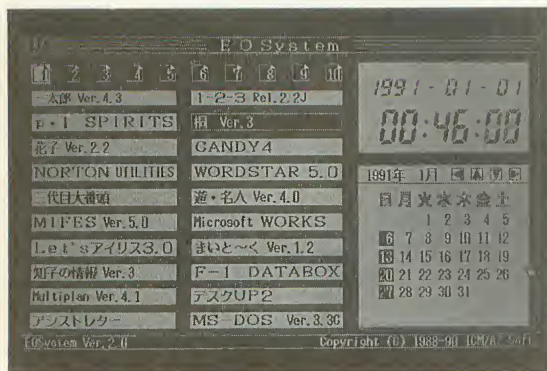
RAMもしくはEEPROMのような半導体メモリに変わるという話をよく聞きますが、半導体メモリに現状の価格低下曲線を大きく下まわると急激な価格低下がない限り、この先5年間でハードディスクとビット・コストが逆転するようなことはなさそうです。

ただし、数Mバイト以下の容量では半導体メモリが使われるようになるでしょう。高記録密度化が著しいハードディスクでは数10Mバイト以下の容量のディスクを作っても安くならないからです。また、耐環境性要求のきつい分野(100Gをはるかに超えるような耐衝撃性、高高度や海底のように気圧が極端に低いまたは高い所での使用、低温もしくは高温での使用など)でもかなり半導体メモリが使われると思われます。

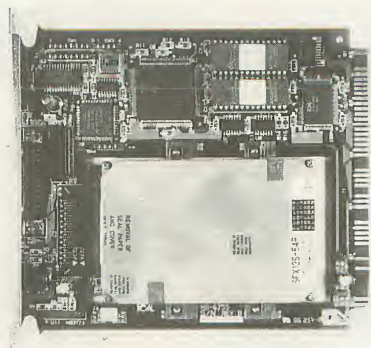
しかし、それ以外の分野ではやはりハードディスクは使い続けられていくでしょう。

## ●ハードディスクのサイズは

次にディスクのサイズはどうでしょうか。



〈写真5〉 操作環境を提供する(EOシステム)



〈写真6〉 2.5インチのハードディスク(C FLAT)





## 第1章

# ハードディスクの機構部

箱の中身はこれほどまでに複雑なのだ

杉本行正/中村正夫/望月将伸/宮崎 修/小寺 勉/渡辺敬明

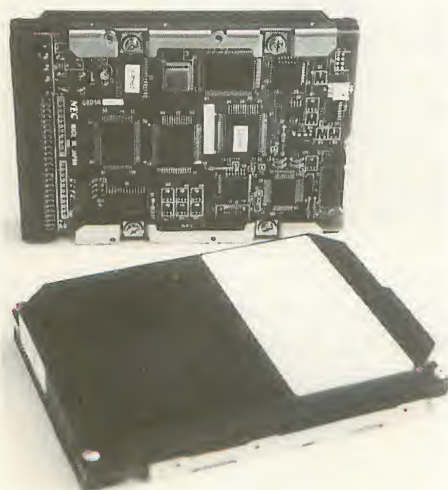
パーソナル・コンピュータ、オフィス・コンピュータなどのOA機器は急速な技術進歩に支えられ、小型化・高性能化が一段と加速されています。それにともない小型・高速・大容量ファイルの需要が急増しており、小型ハードディスクも主役は5.25インチから3.5インチ装置へと移り、今や2.5インチ装置が使用されるまでになっています。

### 機構部

ハードディスク・ドライブの構成は写真1に示すように、機構部とこれらを制御する電子回路部に大別されます。それらの機能ブロック図を図1に、代表的な製品(日本電気)を表1に示します。

#### ● 機構部の構成

機構部は写真2に示すようにベース、カバーから構成されるDE(ディスク・エンクロージャ)内部に磁気ディスク媒体、磁気ヘッド、スピンドル機構、ヘッド・ポジショニング(ヘッド位置決め)機構、塵埃除去機構などが機能的に配置され構成されています。



〈写真1〉3.5インチ・ハードディスク装置  
(上：電子回路部，下：機構部) (D3855 日本電気㈱)

#### ● 媒体

媒体はアルミニウム合金基板の表面に薄い磁性膜を生成したもので、この磁性膜の磁化反転により情報を記憶します。媒体はその寸法、磁性膜生成工法によって図2のように分類されます。大容量化には高密度記録が必要であり、また高密度記録媒体の製造に有利な薄膜型(メッキ、スパッタ)に移行しつつあります。

#### ● ヘッド

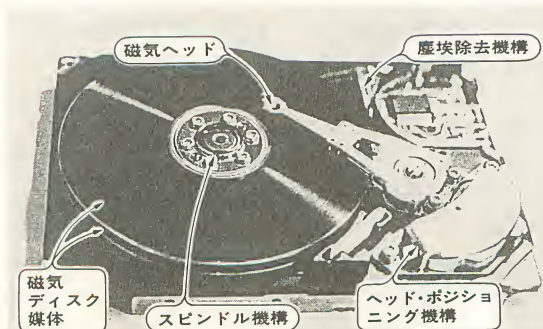
ヘッドはリード/ライト・トランスジューサを備えたスライダ部とそれを支持するサスペンション部から構成されます。媒体が回転することによる運動エネルギーと空気粘性から、スライダ部は媒体表面上をわずかに(サブミクロン：0.1~0.2 $\mu$ m)浮上します。スライダ部を飛行機のボーイング747(約70m)にたとえるなら、高度3mmで飛行しながら尾翼付近に取り付けられたトランスジューサにより、磁化パターンを記録・再生していることに相当します。

媒体の高密度記録化に対応してトランスジューサ部の製法もフェライト→MIG(メタル・イン・ギャップ)→TFH(薄膜ヘッド)と記録・再生効率の向上が図られています。

#### ● スピンドル機構

スピンドル機構は媒体を高精度に回転させる機構で、軸受け部とモータ部により構成されます。

軸受け部は回転精度に大きな影響を与えるため、1対の高精度ボール・ベアリングが使用されます。モータ



〈写真2〉ハードディスク・ドライブの内部

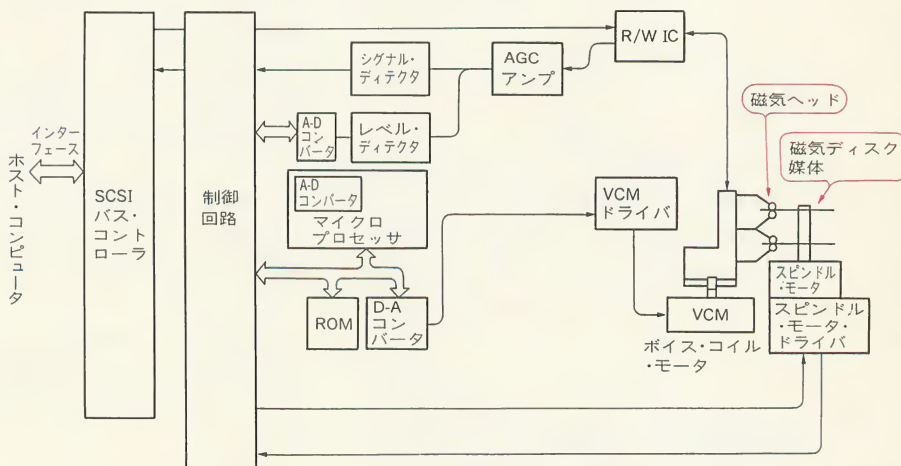


〈表 1〉  
小型ハードディスク  
装置の仕様〔日本電気(株)〕

		3.5 インチ HDD				5.25 インチ HDD	
型 名		D3142	D3×61*	D3×35*	D3×55/56*	D5×62*	D5×82*
記憶容量 (M バイト)	アンフォーマット	53.4	134.45	56.4	131.3	385	765/763
	フォーマット	42.0	118/114	45.0	105.0	300/328	664/649
平均シーク時間 (ms)		28	20	25	25/19	18	16
データ転送速度 (K バイト/s)		625	1250	1500	1500	1250	1875
インターフェース		ST506	ESDI/SCSI/ PC/AT	SCSI/ PC/AT	SCSI/ PC/AT	ESDI/ SCSI	ESDI/ SCSI
外形寸法 (mm)	幅	101.6	101.6	101.6	101.6	146	146
	奥行き	146	146	146	146	208	208
	高さ	41.3	41.3	25.4	25.4	82	82
重量 (kg)		1.0 以下	1.0 以下	0.5 以下	0.5 以下	3.5 以下	3.5 以下

\*: ×=1: ST インターフェース, ×=6: ESDI, ×=7: PC/AT, ×=8: SCSI

〈図 1〉  
ハードディスク装置の  
機能ブロック図



タ部は軸受け部に直結された、長時間使用が可能な  
**DC ブラシレス・モータ**が採用されています。

このモータには従来、回転位置検出にホール素子などが組み込まれていましたが、近年の小型化指向から検出素子のないセンサレス・タイプが採用され始めています。

#### ● ヘッド・ポジショニング機構

これはヘッドを媒体上の目的位置(シリンダ)に位置決めするもので、回転自在のピボットに保持されたアームの一端にヘッドが取り付けられ、他端に駆動モードが取り付けられています。

従来の駆動モータはステッピング・モータ、DC トルク・モータなどが使用されていましたが、アクセス・タイムの短縮のため低慣性高トルク型のボイス・コイル・モータが採用されています。

また、そのボイス・コイル・モータの心臓部ともいえる磁石には現在入手可能な最高性能の永久磁石である **Nd-Fe-B 系** が使用されています。

#### ● 塵埃除去機構

ヘッドの浮上量は前述したように非常に微小であり、目に見えない微細な塵埃でもヘッドと媒体の間に入り込むと媒体が損傷し、記録された磁化パターンが失わ

〈図 2〉 寸法と製法による媒体の分類

寸 法	5.25 インチ (130×40×1.905mm)
	3.5 インチ ( 95×25×1.27mm)
	2.5 インチ ( 65×20×0.89mm)
製 法	塗布型 (磁性粉をバインダに溶かし塗装)
	薄膜型 (磁性体を直接メッキまたはスパック)

れることになります。したがって DE 内部は高潔浄度に保つ必要があり、万一の外部塵埃やわずかな内部発塵に備え、塵埃除去機構としてエア・フィルタが設けられています。

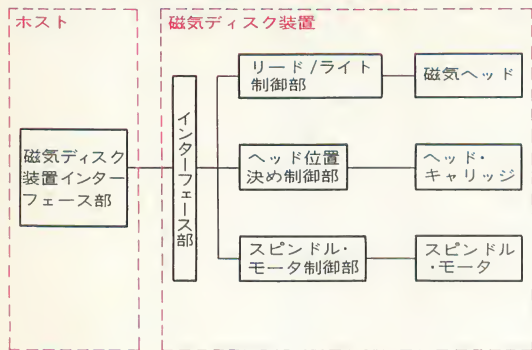
### 電子回路

小型ハードディスク・ドライブの電子回路は、1 枚のプリント板の上に渾然一体となって配置されています。図 3 に示すようにその機能により**インターフェース部**、**スピンドル・モータ制御部**、**ヘッド・ポジショニング制御部**、**リード/ライト部**に大別されます。

#### ● インターフェース回路

インターフェース部は磁気ディスク・コントローラ

〈図3〉磁気ディスク装置の構成



もしくはホスト(パソコン本体など)と接続するための回路です。

5.25 インチ以下のディスクで磁気ディスク・コントローラと接続するタイプとしては **ST506/ST412 インターフェイス** および **ESDI** が一般的です。

これに対して最近では磁気ディスク・コントローラの LSI 化が進み磁気ディスク装置の中にコントローラを取り込むタイプが主流になってきました。このようなタイプのディスクはエンベデッド・ディスクと呼ばれ、PC/AT インターフェースと SCSI が一般的です。

エンベデッド・ディスクのインターフェース部の構成を図4に示します。ホスト(パソコン本体など)からデータの記録や読み出し方法を指示するコマンドの解釈、ホストとのデータの送受信および媒体上とのデータの送受信を行うバッファRAM制御、コマンドの実行結果の処理・報告を行う部分です。

ここでのリード/ライト制御信号などは ST506 や ESDI の信号と同等の信号で、逆にいえば ST506 や ESDI のインターフェースの後にコントローラがつながった形になっています。

▶ ST506/ST412 インターフェース

1979年にシーゲート社からマイクロコンピュータ用のディスクとして5.25インチ・サイズのディスクST506が発売されました。

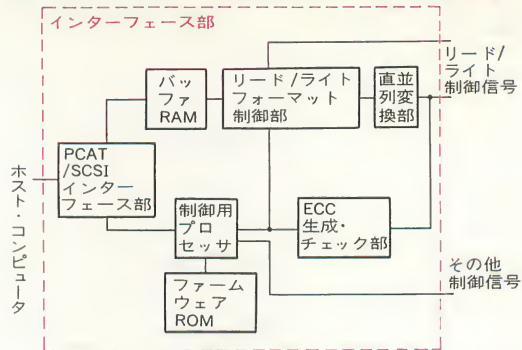
ST506 はフロッピー・ディスク・インターフェースのデータ・レート (500Kbps) と回転速度を 10 倍 (5Mbps) に、トラック数とヘッド数をそれぞれ倍にしたインターフェースです。このような関係からフロッピー・ライク・インターフェースとも呼ばれます。

ST506 のシークはフロッピー・インターフェースの場合と同様、コントローラからのステップ・パルスによって直接動作します。

したがってシーク・スピードはコントローラからの  
ステップ・パルスに依存するため、ドライブの性能が  
向上しても、それを反映することができません。

そこでシーゲート社はつぎの ST412 で倍トラック

〈図4〉エンベデッド・ディスクのインターフェース部の構成



化(306 シリンド)と同時に磁気ディスク・ドライブの上に CPU をのせ、**バッファード・シーク**と呼ぶモードを追加しました。これはシーク・パルスをもとめて送っておき、ドライブ側で最適になるように実際のシーク動作を独自に行うものです。

このバッファード・シークまでを含めたインターフェースを ST506/ST412 インターフェースもしくは単に ST506 インターフェースと呼びます。

▶ ESDI(エンハンスド・スモール・デバイス・インターフェース)

ST412 はシリンダ数 306 と ST506 の倍のシリンダを持ちます。ディスクの容量を増やすために ST506 インターフェースではシリンダ数とヘッド数を増やしました。

しかし、データ・レートはディスク・コントローラに依存するため 5Mbps の MFMM のまま変更されませんでした。また、MFMM 信号でインターフェースされ、データ・セパレータ (MFMM → NRZ 変換) はコントローラ側にありました。したがって、データ・レートが上昇するとインターフェース部の信号劣化が問題になります。

そこで、ESDI ではデータ・セパレータをドライブ側に置き、NRZ データとクロックでインターフェースすることで10Mbps から15Mbps のデータ・レートまで扱えるようにしました。

また、シーク・パルスを送出する時間的なロスをなくし、その他のコマンドがやりとりできるようにシリアル・ラインでコマンド転送を行うようにしました。

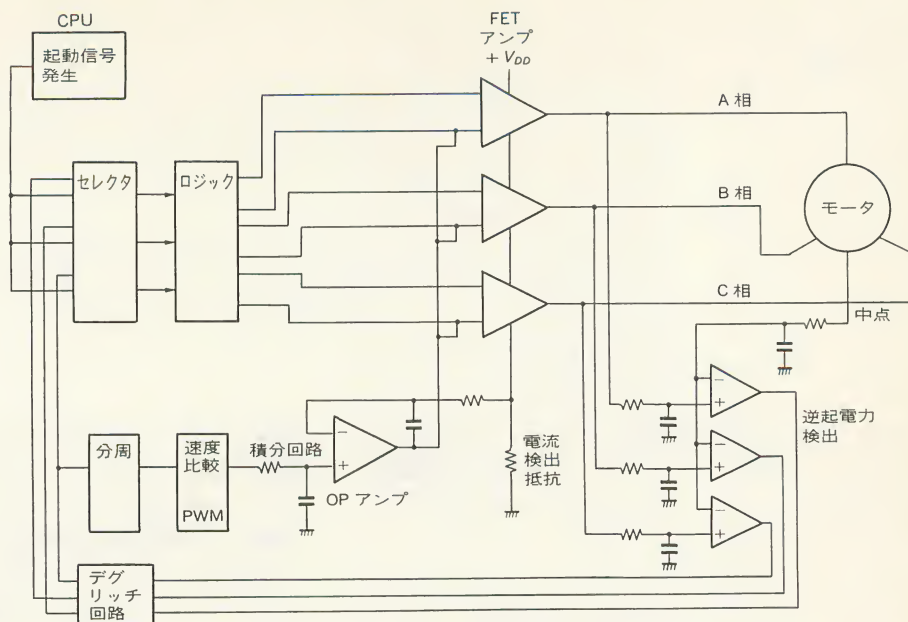
## ▶ PC/AT インターフェース

ハードディスクに使用されている PC/AT インターフェイスは IBM のパソコン PC/AT の増設スロットに使用されているインターフェイスをハードディスク・ドライブ用に小型化したもので、ATA(ATパス・アタッチメント)として仕様化・標準化が進められています。

このインターフェースには現在のパソコン CPU の主流である 80286/386 に少ない部品を追加するだけで



〈図5〉  
スピンドル・モータ  
回路のブロック図



ホスト回路が構成できるという特徴があり、このインターフェースがパソコンに標準装備されることでパソコンのハードディスク内蔵比率は一層高まるものと思われる。

#### ▶ SCSI インターフェース

SCSI(スモール・コンピュータ・システム・インターフェース)は従来から使用されてきた SASI の機能拡張と仕様統一を目的として米国の ANSI で制定され、現在では一層の機能拡張を行った SCSI2 の製品化が開始されています。

このインターフェースは十分な検討のもとに仕様化されており、磁気ディスク装置以外にも光ディスク装置、スキャナ、プリンタなどの機器を接続できる特徴があり、システム機能拡張には最適のインターフェースです。

インターフェース部には上記に示した機能のほかにエラー発生時のリトライ、ECC による修正およびバッファ RAM のデータ管理などの役割があります。エラー検出・訂正には ECC コードが使用されますが、そのコードも 4 バイト ECC から現在は 7 バイト ECC になり、さらに性能を向上させたリード・ソロモン・コードの使用も始まっています。このほか読み出しと同時に修正が行える方式も使われ始めています。

#### ● スピンドル・モータ制御回路

ディスク・ドライブのスピンドル・モータとしては前述した DC ブラシレス・モータが使用されています。この回路(図5)の機能はモータを短時間で起動させることと高精度で回転させることにあります。

モータの回転速度は現在ではデジタル回路で構成される速度検出回路から得られる速度信号と基準信号

との差として得られる誤差信号により、モータ駆動電流を制御するフィードバック制御で行われます。

スピンドル機構のところで説明したように、小型化のために位置検出素子を省いたセンサレス・タイプのモータが使用されたため、モータ駆動には回転位置検出機能が必要となります。

通常はモータの回転によってモータ・コイルに発生する逆起電圧を検出することにより回転位置を検出しています。

モータが回転している場合には逆起電圧が発生するため上記の方法で回転位置は検出することができますが、起動時には逆起電圧が発生していないため簡単には位置検出ができず、モータが逆回転するなどの欠点があります。

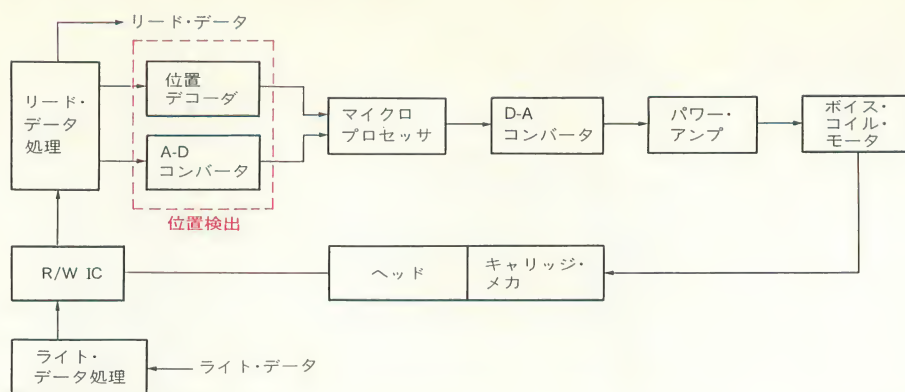
実際のディスク・ドライブではこの欠点を克服するためにモータが回転しない程度の短時間のモータ駆動を行った際のモータ・コイルに発生する電流または電圧をマイクロプロセッサで処理するなどして、起動時の回転位置を検出するといった工夫がなされています。

磁気ディスク装置の高速・大容量化の一手法としてディスク・アレイの開発が進められています。その効率向上のためには複数のディスク・ドライブを同期して回転させる必要があります。このスピンドル同期機能をサポートしたハードディスク・ドライブも出現しています。

#### ● ヘッド・ポジショニング制御回路とサーボ

図6にヘッド・ポジショニング回路を示します。これは前述したヘッド・ポジショナ部の制御を行うもので、ヘッドを目標トラックに移動させ、目標トラック中央に保持する回路です。このような位置や速度を目

〈図 6〉  
ヘッド・ポジショ  
ニング回路



標値に追従させる制御回路を一般にサーボ回路といいます。ハードディスク・ドライブではヘッドを移動させる動作を**シーク**、ヘッドを保持する動作を**トラック・フォロー**といいます。

#### ▶ セクタ・サーボ

平均シーク・タイムが 25ms 以下の高速タイプのハードディスク装置はヘッドを動かす動力源であるアクチュエータにボイス・コイル・モータを使い、サーボ方式として専用サーボ面からサーボ情報による**デディケイテッド・サーボ方式**と、データ面にサーボ情報を埋め込んだ**エンベデッド・サーボ方式**が使われています。

ここでは、最近の小型ドライブに多用されているエンベデッド・サーボ方式のひとつである**セクタ・サーボ**について説明します。

セクタ・サーボ方式は通常 512 バイトのデータと ID 情報などで構成されるセクタの間に、位置情報を記録したサーボ・セクタを埋め込む方式です。この方式では位置情報が媒体 1 回転につきセクタ数(30~70 回)しか得られないため、アナログ制御よりマイクロプロセッサによるデジタル制御が適しています。

シーク制御は目標までの距離の関数としてあらかじめ設定した速度パターンと実際の速度を比較し、その差に比例した電流をボイス・コイル・モータに流すことによって速度を目標値に追従させます。

#### ▶ 速度を位置情報やモータの電流値から算出

サーボ・セクタから得られる情報は位置だけなので、速度は位置情報やボイス・コイル・モータの電流値から算出します。デジタル・サーボでは主として最新のセクタの位置情報とひとつ前のセクタの位置情報の差分で速度を計算します。

こうして求めた速度は全セクタでの速度と最新セクタでの速度の平均速度であり、およそセクタ周期の 2 分の 1 の時間遅れがあります。しかし、現代制御理論のオブザーバを使って速度を推定し、遅れを回避しています。

#### ▶ トラック・フォロー

トラック・フォロー中は位置の誤差をなくすように

制御します。この制御ループを安定化するために位相補償が必要で、デジタル・サーボではマイクロプロセッサの演算処理でデジタル PID 制御や現代制御理論による動的補償器を構成して安定化します。

大容量化には狭トラック化が必要であり、現在のディスク・ドライブのトラック幅は 10 $\mu$ m 前後となっています。このような狭トラックのディスク・ドライブでは極めて高精度のトラック・フォローが要求されます。

セクタ・サーボ方式の場合にはデータの読み書きに使用するヘッド自身で位置を検出するために、温度差によるヘッド間のトラック・オフセットを無視できるという特徴があります。説明を省略したデディケイテッド・サーボ方式の場合もトラック・フォロー時にはセクタ・サーボを一部併用する**ハイブリッド・サーボ方式**が採用されつつあります。

今後はアナログ・サーボを上回る現代制御理論を活用したデジタル制御化と回路の LSI 化がさらに進展すると思われます。

### ● リード/ライト制御回路

この部分はホストから転送されたデータをヘッドを介して媒体に書き込むライト部とヘッドを介して媒体の信号を読み出しホストに転送するデータ形式に変換するリード部から構成されます。各部の機能を図 7 にしたがって説明します。

#### ▶ 変復調部

変調部は NRZ 形式のデータを磁気記録する際、より多くの情報量を書き込めるデータ形式に変換する部分です。変調方式は **FM → MFM → RLL(2,7) → RLL(1,7)** と高効率化が図られてきています。

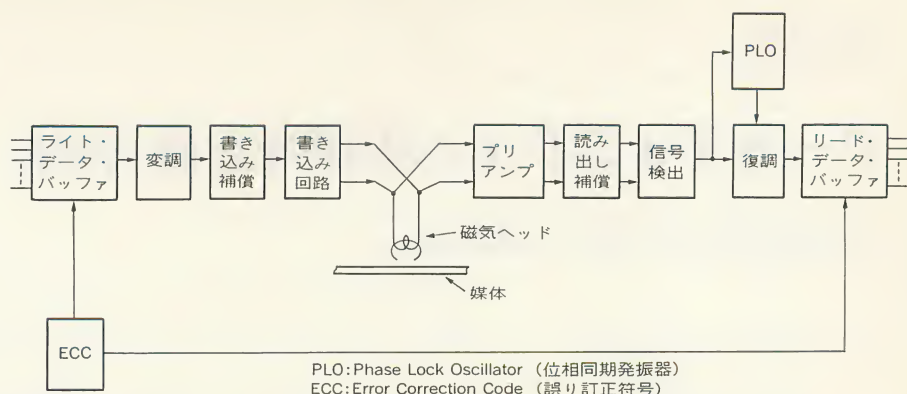
復調部は上記の変調方式で変調された読み出し信号を NRZ 形式に戻す部分です。

#### ▶ リード/ライト補償部

書き込み補償部は高密度記録の際に発生する磁化反転干渉による読み出し信号のピーク・シフトに対して書き込み時の磁化反転タイミングの微少補正を行う部分です。



〈図 7〉  
リード/ライト回路  
のブロック図



読み出し補償部は読み出し時の波形等価によるピーク・シフトの減小化を行う回路です。

#### ▶ データ・セパレータ

データ・セパレータ部はパルス信号に整形した読み出し信号を基にリファレンス・クロックと呼ばれるリード処理時の基本クロックを作成するとともにリード・データのリファレンス・クロックへと同期化を行います。

リード/ライト回路の構成は小型ハードディスク・ドライブでは回路のLSI化が進行し、現在では2~3チップ構成になってきています。近い将来には1チップLSIも現れると思われます。

以上、小型ハードディスク装置の概要を説明しました。技術面からその特徴を見ると、機械工学・制御工

学・電子工学・システム工学などが複雑に交錯しながら集大成されています。

またLSIやマイクロプロセッサなどの利用によって、変調方式やエラー処理技術など高度な情報理論を応用したインテリジェント化が進んでおり、高信頼性維持に大きく貢献しています。

磁気ディスク装置は半導体メモリ、光ディスク装置の開発により、その製品寿命が幾度となく論じられてきましたが、予想を上回る記録密度の向上により大容量化・小型化・高速化を達成し、スーパーコンピュータ用からパーソナル・コンピュータ用まで多彩な製品を創出しており、今後とも発達し続けるものと考えられます。



## フロッピー・ディスク装置のすべて

FDD全タイプの基礎から応用まで

高橋昇司 著

A5判 352頁 定価2,500円(税込)

### 目次

- 1 フロッピー・ディスク装置の開発動向 フロッピー登場の歴史/2インチや高密度大容量FDD/垂直磁気記録方式など
- 2 FDDの要素技術 各種ヘッドとその特徴/ヘッド支持機構/ディスク駆動機構/ディスク保護機構/その他の主要構成部品など
- 3 インターフェース FDD信号インターフェース/信号レベル/電源インターフェースなど
- 4 互換性技術 構造的な互換性問題/電磁変換特性関連の互換性/高密度・低密度間の互換技術など
- 5 FDおよびFDフォーマット FD規格/トラック・フォーマット/コピー・プロテクト技術など
- 6 高記録密度化技術 FM・MFM記録方式/記録再生の原理/書き込み補償/波形等価方式/データ・セパレータ/タイム・マージン/M<sup>2</sup>FM・GCR・

- RLLCなど
- 7 FDDの評価技術 電磁変換特性/機能特性/互換性特性/環境特性/信頼性・耐久試験
- 8 FDD制御用LSI FDC LSIの動向/FD1791ファミリ/μPD765ファミリ
- 9 データ・セパレート回路 VFOLSIの動向/SED 9420/TC8568M/HA 16632 AP/μPD 71065 G・71066 CT/ディジタル・データ・セパレータ/FDC 9216・9229/CXD1190P
- 10 FDCとFDDのインターフェース MB8876とのI/F/μPD765AとのI/F
- 11 FDD使用上の注意事項 エラー処理/ノイズ対策/防塵対策と冷却/システム・マージン測定法/ヘッドとメディアの吸着/電池駆動など

CQ出版社

## 第2章

# ハードディスク上への信号記録/再生

記録/再生もカセットみたいに単純ではない

松村清明

デジタル・データを磁気ディスクに記録し、それを再生してもとのデジタル・データを得る方法について述べます。

### 概要

記録再生系の概念を示したブロック図を図1に、また対応する各部の信号パターンを図2に示します。

HDC(ハード・ディスク・コントローラ)から出力されるNRZ(デジタル・データをそのまま0, 1の直列信号にしたもの)信号(a)を変調(b)し、このパルスをも1/2分周して、パルスごとに書き込み電流方向を反転し(c), この電流を磁気ヘッドに与えて磁化パターン(d)をディスク上に作ります。デジタル磁気記録では通常このような磁化反転によって、データを記録します。

次に磁気ディスク上に書かれた磁化パターン(d)を磁気ヘッドで読み出すと(e)の再生波形が得られます。実

際には図の波形より時間軸方向右にずれた波形が磁気ヘッドから出力されますが、ここでは説明のため磁化反転位置と再生波形のピークを揃えました。(f)、(g)の波形も同様にディレイがない場合の波形を示しています。

この再生波形を整形し、ピークを検出することで検出パルス(f)が得られます。このパルスを基にPLLで記録時のクロックを再生し、復調回路によって復調データ(g)を得て、HDCへ入力します。

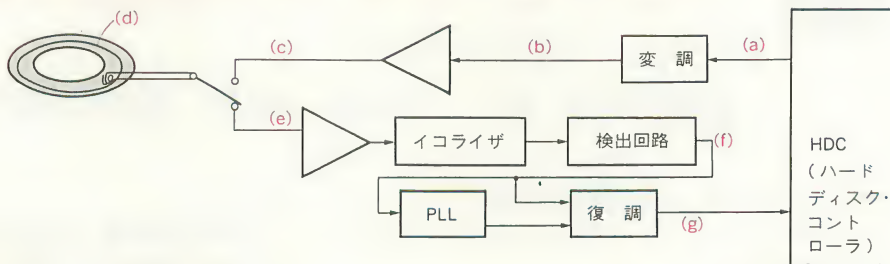
これらの回路を総称して記録再生系もしくは信号処理系と呼びます。

以降の項で各ブロックについて説明します。

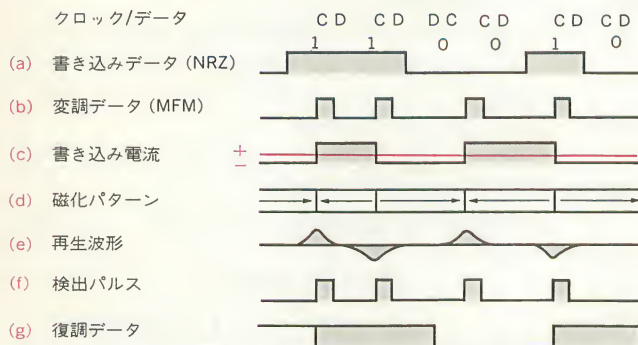
### 変復調方式

デジタル・データを記録媒体に記録できるような形に変換することを変調といい、その記録をもとのデ

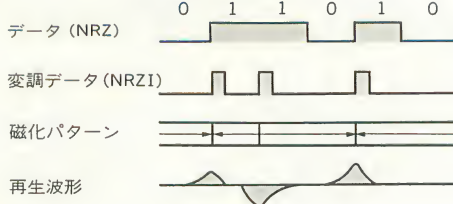
〈図1〉  
記録再生系のブロック図



〈図2〉 各ブロックにおける波形



〈図3〉 NRZI方式





ータに戻すことを復調といいます。

### ● NRZI 方式

最も単純な変調方式である **NRZI 方式**を図3に示します。

NRZI(Non Return Zero Inverted)方式は1のときに磁化方向を反転し、0のときはそのままにする方式です。

この方式は単純ですが、0が連続した場合には磁化反転が発生しません。したがってそんな場合には**再生波形から記録時のクロックをつくりだすこと(セルフ・クロッキング)**ができません。

そこで、一般的には別にクロックを記録しておき、その信号を別のヘッドで読み出してクロックを作ります。しかし、この方法ではクロックの位相とデータの位相が合っていることが必要になります。

ビット・レートが低いときにはNRZI方式を変調方式として使用したディスクもありましたが、ビット・レートの上昇とともに、データとクロックの同期が困難になり、セルフ・クロッキング可能な(再生波形からデータとともにクロックが再生できる)**FM(Frequency Modulation)方式**や**MFM(Modified Frequency Modulation)方式**が開発されました。

### ● FM 方式と MFM 方式

図4にこれらの方式を示します。データ/ビットはクロック・ビット(図中のC)とデータ・ビット(図中のD)の2ビットに変換されます。FM方式はすべてのクロック・ビットでパルスが発生させ、データ・ビットではデータが1の場合のみパルスが発生させる方法で、クロック・ビットが必ずあるため再生波形からクロックを容易に再生できます。しかし、1ビットに対して2回磁化反転が発生するので、**最小磁化反転間隔**がNRZI方式の1/2になってしまいます。

MFM方式は磁化反転間隔を落とさずにセルフ・クロッキングができる方式として開発されました。この方法はデータが1の場合にはデータ・ビット位置にパルスが発生し、0が2ビット以上連続した場合には2ビット以降の0のクロック・ビット位置にパルスが発生します。これにより、最小磁化反転間隔はデータ・ビットの間隔となりNRZI方式と等しくなります。ただし、クロック・ビットの発生が0が連続した場合に限られるため、クロック・ビットのみからクロックを再生することはできず、データ・ビットを含めデー

タ・レートの倍のレートで同期クロックを発生する必要があります。

そこで、このクロックをデータ・ビットに同期したタイミングで1/2分周する必要があります。そして図5に示すように**MFMの変換方式では発生しないようなビット列(ミッシング・クロック)**をマークとしてビット同期を取ります。またこのマークは後述するデータ・フォーマットの中のアドレス・マークとして使用され、バイトの同期を取るためにも使用されます。

MFM方式は最小磁化反転間隔が大きい割に変換方式やクロック再生が容易であるため、フロッピー・ディスクや小型ハード・ディスクで多数使用されています。

### ● RLL 方式

ハード・ディスクの高容量化にともない固定ディスクのようにメディア互換性を必要としない用途においては、より高密度な変換方式を使用するようになってきました。そこで開発されたのが**RLL(Run Length Limited)方式**です。RLL方式には多くの方式が提案されていますが、現在最も多く使用されているのが**RLL(2, 7)**で、多く採用されつつあるのが**RLL(1, 7)**です。

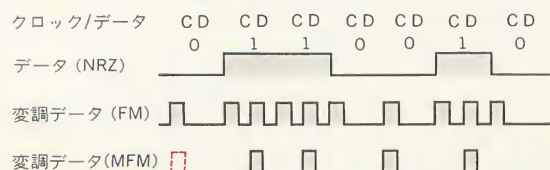
ここで、RLL(x, y)としたときのxとyは**コード化されたときの1と1の間に出現する0の数の最小と最大**です。この表現を用いればMFMはRLL(1, 3)ということになります。これらのデータとコードの変換表を表1に示します。またRLL(2, 7)の変換例を図6に示します。

MFMの場合には0が始まるときを除いてデータ1ビットがコード2ビットに対応します。しかし、RLL(2, 7)の場合にはデータの並びによって2ビットもしくは3ビットが、4ビットもしくは6ビットのコードに対応し、RLL(1, 7)においては先行するビットも含めると2ビットから5ビットのデータをもとに1.5倍にコード化されます。

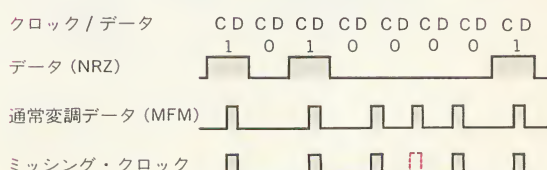
このことは再生データにひとつの誤りがあった場合、復調において複数のビットの誤りとして出現する(エラーが伝播する)ことを意味します。したがって、MFM, RLL(2, 7), RLL(1, 7)の順にエラーの伝播ビット数が多くなります。

RLL(2, 7)の最小磁化反転間隔はMFMの1.5倍であるため、**線記録密度(磁化反転密度)**を一定するとMFM方式で記録する場合と比較して1.5倍のデータ

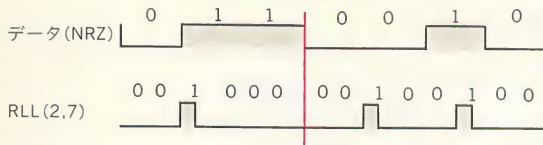
〈図4〉 FM, MFM 方式



〈図5〉 ミッシング・クロック



〈図 6〉 RLL 変換例



を記録できます。ただしそれだけ**符号の冗長度**が少ないので、ミッシング・クロックのようなビット同期方式は使えません。

そこでソフト・セクタ(第3章で解説)においてはセクタ・マークとして24ビットから32ビット程度の**イレーズ・ギャップ**を作り、これを再生パルスのギャップから検出する方法が一般的です。またシンク領域においてビット同期し、アドレス・マークのデータでバイト同期します。

各種変調方式のパラメータを表2に示します。磁化反転間隔は、先に述べたものをまとめて表しています。**検出窓幅**はビット列の位置を検出するための窓の幅で、大きい程復調が容易であることを示します。NRZI方式はセルフ・クロッキングできないので例外です。

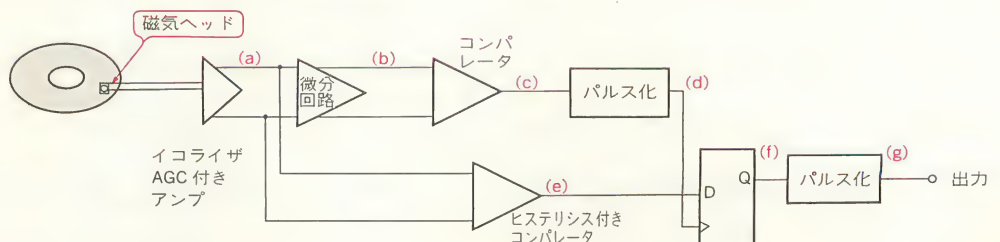
MFM方式とRLL(2, 7)はデータ1ビットあたりコード2ビットであるため、どちらも $0.5T$ の窓幅になります。これに対し、RLL(1, 7)の磁化反転間隔は $1.33T$ でRLL(2, 7)の $1.5T$ より若干悪くなりますが、RLL(1, 7)はデータ2ビットがコード3ビットになるので $2/3$ の窓幅を持ち、復調が容易であるといえます。これが最近高データ・レートのディスクでRLL(1, 7)の採用が多くなってきた理由です。しかし、先に述べたようにエラーの伝播が大きいため、RLL(2, 7)を採用し続けるメーカーも多いようです。

## 信号検出

磁化パターンを磁気ヘッドで読み出し、パルス化するまでの信号検出部分のブロック図を図7に示します。対応する各点の信号波形を図8に示します。

磁化パターンを磁気ヘッドにより読み取り、再生した波形(a)のピークが、磁化パターンの変化点を表します。そこでこれを微分(b)し、コンパレータ(c)で0クロス点をエッジとして取り出します。このエッジをパル

〈図 7〉  
信号検出部  
ブロック図



〈表 1〉 RLL 方式変換表

方式	データ	コード
MFM [RLL(1,3)]	0	×0
	1	01
RLL(2,7)	000	000100
	10	0100
	010	100100
	0010	00100100
	11	1000
	011	001000
	0011	00001000
RLL(1,7)	01	×00
	10	010
	11	×01
	0001	×00001
	0010	×00000
	0011	010001
	0000	010000

×=先行するビットの補数

〈表 2〉  
各種変調方式の  
パラメータ

方式	磁化反転間隔		検出窓幅
	最小	最大	
NRZI	$T$	$\infty$	$T$
MFM	$T$	$2T$	$0.5T$
RLL(2,7)	$1.5T$	$4T$	$0.5T$
RLL(1,7)	$1.33T$	$5.33T$	$0.67T$

ス化してパルス列(d)を得ます。

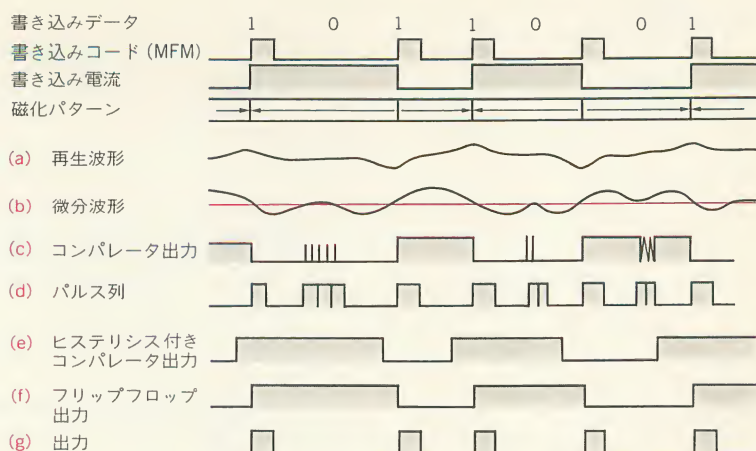
ここで図8にあるように磁化反転間隔が大きい場合、再生波形のピークとピークの間に振幅のないタイミングができます。これを微分すると微分波形はゼロ近くでノイズ性の信号となり、コンパレータ出力にノイズが発生し、疑似パルスの発生をまねきます。

このノイズを除く方法は種々考案されていますが図7、図8では**原波形スライス方式**と呼ばれる回路を説明しています。

再生波形(a)をヒステリシスのあるコンパレータに通してコンパレータ出力(e)を得ます。この信号は書き込み電流波形を少し早めたような波形となります。これを(d)パルス列でサンプリングすると、真のパルスの中間あたりにある疑似パルスではヒステリシス・コンパレータ出力が直前の真のパルスのときと同相であるため、(f)のような波形が得られます。これをパルス化して必要な波形(g)を得ることができます。



〈図8〉 信号検出部の波形



## 復調回路

信号検出部から出力されたパルス列を HDC に入力するために NRZ に変換する部分を**復調回路**といいます。この部分のブロック図を図9に示します。

復調回路は大きく **PLL** (Phase Locked Loop) とデコーダに分かれます。PLL の部分はリード・データと位相同期した VCO クロックを作ります。このクロックはリード・データのサンプリング・クロックになります。

ディスク用の PLL のロックする周波数はディスクの回転誤差である  $\pm 1\%$  ぐらいの精度で一定周波数です。重要なことは ID 部やデータ部の前のシンク領域

(3章)で、リード・データと VCO の位相差を急速に引き込むことと、データ領域でゆるやかな周波数変動に追従することです。変調方式によって種々の方法がありますが、PLL についてはここでは述べません。

ハード・ディスク用 VFO チップの仕様書などを参照して下さい。

デコーダの部分は変調方式によってかなり異なります。図10に MFM の場合の、図11に RLL(2,7) の場合のビット・シンクおよびデコードのブロック図を示します。

図10の MFM の場合には、VCO クロックで同期化された MFM データをサンプリングし、ミッシング・クロックによりつくられたユニークなアドレス・マークを検出します。これは、VCO クロック(データ・

# トランジスタ技術 SPECIAL No.11

好評発売中!

B5判 176頁  
定価1,540円(税込み)  
送料 260円

## 特集 フロッピー・ディスク・インターフェースのすべて 需要の急増するFDDシステムの基礎から応用

### (CONTENTS)

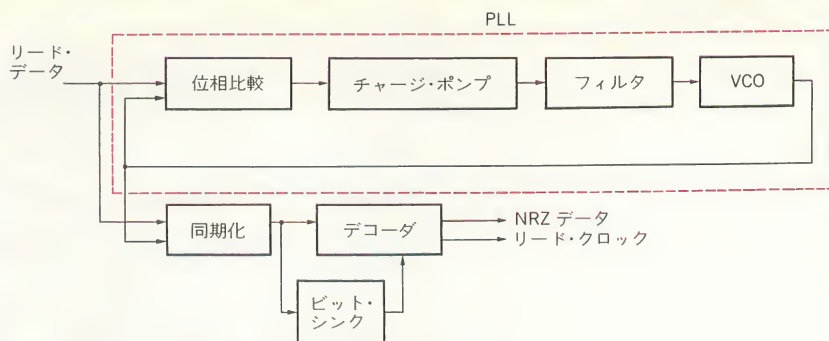
- 第1章 フロッピー・ディスク・システムの基礎
- 第2章  $\mu$ PD765Aを用いたフロッピー・ディスク・インターフェース
- 第3章 MB8877Aを用いたフロッピー・ディスク・インターフェース
- 第4章 MB89311Aを用いたフロッピー・ディスク・インターフェース

- 第5章 WD2793を用いたフロッピー・ディスク・インターフェース
- 第6章 PC9801のフロッピー・ディスク・インターフェースの詳細
- 第7章 PC9801のFD-BIOSを解析する
- 第8章 PC9801へのFDDの接続法と2DDから2HDへの改造

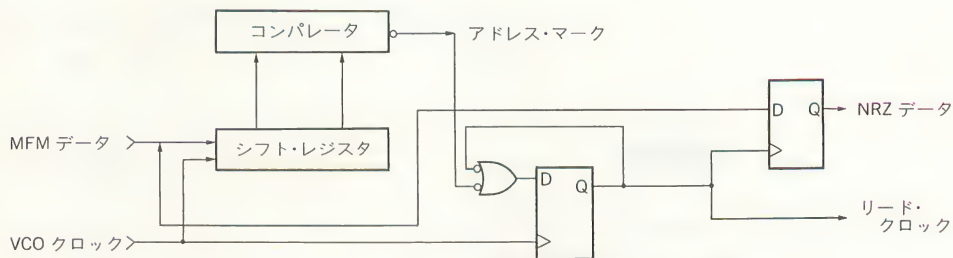
CQ出版社 ㊟170 東京都豊島区巣鴨1-14-2

振替 東京0-10665

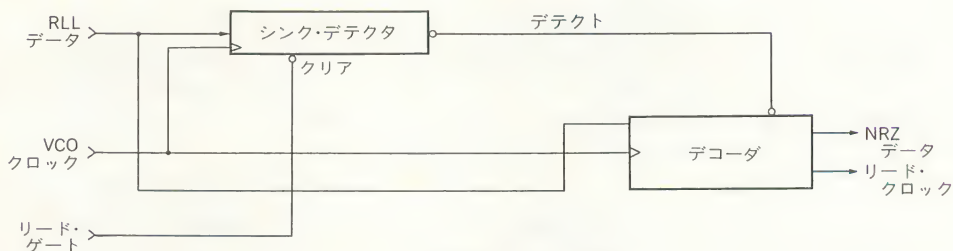
〈図 9〉 復調回路



〈図 10〉  
MFM のビット・シンク



〈図 11〉  
RLL (2,7) の  
ビット・シンク



クロックの倍)で MFM データをサンプリングすることで、一意に検出できます。検出ビットにより VCO の 1/2 分周を調整し、リード・クロックを作ります。

また MFM の場合には 1 のデータに対して、必ずビットが立っているので、MFM データを単純にリード・クロックでサンプルすることにより、NRZ データをつくることができます。

図 11 の RLL (2,7) の場合はもう少し複雑です。まず VCO クロックに同期化された RLL データとともにシンク領域と同時に立ち上がるリード・ゲート信号を使います。リード・ゲートが立ち上がって PLL がロックすると RLL データはシンク・パターン (0001 もしくは 001 の連続) になります。一定回数 (たとえば 8 回) 連続してシンク・パターンが入力されるとシンク・デテクタはシンクを検出したということでデテクト信号を出力します。

この信号に同期してデコーダはリード・クロックを

ビット同期し、直前 4 ビットの RLL データをデコードして NRZ 信号をつくります。

以上述べた各ステップにより磁化データをリードして NRZ 信号を復調し、書き込み時に HDC から出された NRZ 信号と同等の信号列を得ます。

#### 参考・引用文献

- (1)\*Max Roth: コストを抑えて磁気ディスクの容量を増やすためには効率の良い信号符号の選択が重要, 日経エレクトロニクス, 1985.7.15, pp.289-297.
- (2) 伊藤陽之助他: デジタル磁気記録技術, トリケップス, 1986.
- (3) National Semiconductor, DP8462 2, 7Code Data Synchronizer.
- (4) WESTERN DIGITAL, WD10C23, 1989.
- (5) National Semiconductor, DP8469 Synchronizer, 1989.
- (6) Adaptec, inc, AIC-270, 1986.
- (7) 日立, HD153009 磁気ディスク用 2-7 符号変復調 IC, 1987.
- (8) 日立, HD153014F データセパレータ, 1990.



# 代表的な HDC

松村清明

## ● $\mu$ PD72061 (日本電気)

$\mu$ PD72061 は  $\mu$ PD7261 を CMOS 化したハードディスク・コントローラ (HDC) です。ハード・セクタ・ディスク (SMD インターフェース)、およびソフト・セクタ・ディスク (ST506 インターフェース) の両方を取り扱うことが可能です。

### ▶ 特 徴

- ・  $\mu$ PD7261A/B とコンパチブル
- ・ CMOS
- ・ 2 種のディスク・ドライブにインターフェース可能  
ハード・セクタ・ディスク (SMD インターフェース)  
ソフト・セクタ・ディスク (ST506 インターフェース)
- ・ プログラマブル・トラック・フォーマット  
データ長: 128~4095 バイト/セクタまで指定可能  
各種ギャップ長可変
- ・ データ転送速度  
24Mbps:  $\mu$ PD72061-24 (ハード・セクタ)  
12Mbps:  $\mu$ PD72061-12 (ハード・セクタ)  
 $\mu$ PD72061-24 (ソフト・セクタ)  
6Mbps:  $\mu$ PD72061-12 (ソフト・セクタ)
- ・ ディスク・ドライブ: 8 台まで接続可能 (SMD インターフェース)  
4 台まで接続可能 (ST506 インターフェース)
- ・ パラレル・シーク動作可能
- ・ マルチ・セクタ機能
- ・ マルチ・トラック機能
- ・ データ・スキャン, データ・ベリファイ機能
- ・ CRC 発生, チェック機能
- ・ ECC 発生, チェック, 訂正機能  
11 ビットまでのバースト・エラーを訂正可能
- ・ DMA データ転送

## ● $\mu$ PD7262 (日本電気)

$\mu$ PD7262ESD (Enhanced Small Device Interface Controller) はシリアル・モード ESDI のハードディスク・ドライブを 7 台まで制御可能なハードディスク・コントローラです (図 A)。

### ▶ 特 徴

- ・ シリアル・モード ESDI 対応
- ・ 7 台までのドライブを制御可能
- ・ ハード・セクタ, ソフト・セクタ混在制御可能
- ・ プログラマブル・トラック・フォーマット

データ長可変: 128~65536 バイト/セクタ

ギャップ長可変: インタセクタ・ギャップ

PLL シンク

スピード・トレランス・ギャップ

- ・ セクタ開始信号選択  
セクタ・パルス/AMF 信号
- ・ データ転送レート  
max. 18Mbps:  $\mu$ PD7262-18  
max. 12Mbps:  $\mu$ PD7262-12
- ・ 23 種類のディスク・コマンド
- ・ パラレル・シーク動作
- ・ マルチ・セクタ/マルチ・トラック/マルチ・シリンダ機能
- ・ インブライド・シーク機能
- ・ データ・スキャン/ベリファイ機能
- ・ エラー・セクタのデータ読み出し機能
- ・ ECC/CRC 選択可  
ECC: 32/56 ビット  
CRC: 16 ビット
- ・ N チャネル MOS

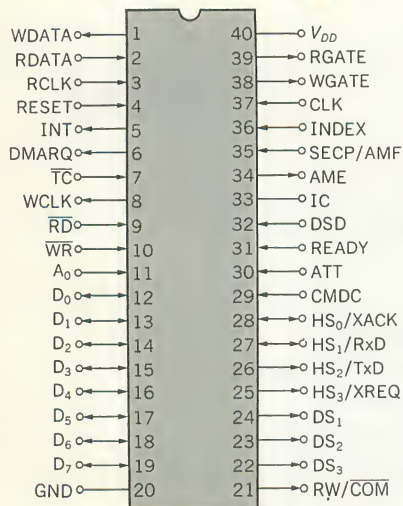
## ● HD63463 (日立)

HD63463 は 16 ビット・プロセッサ HD68000 の周辺 LSI として開発された CMOS の HDC です。SRAM を内蔵し、低速インターフェースしか持たないホストにも接続できます (図 B)。

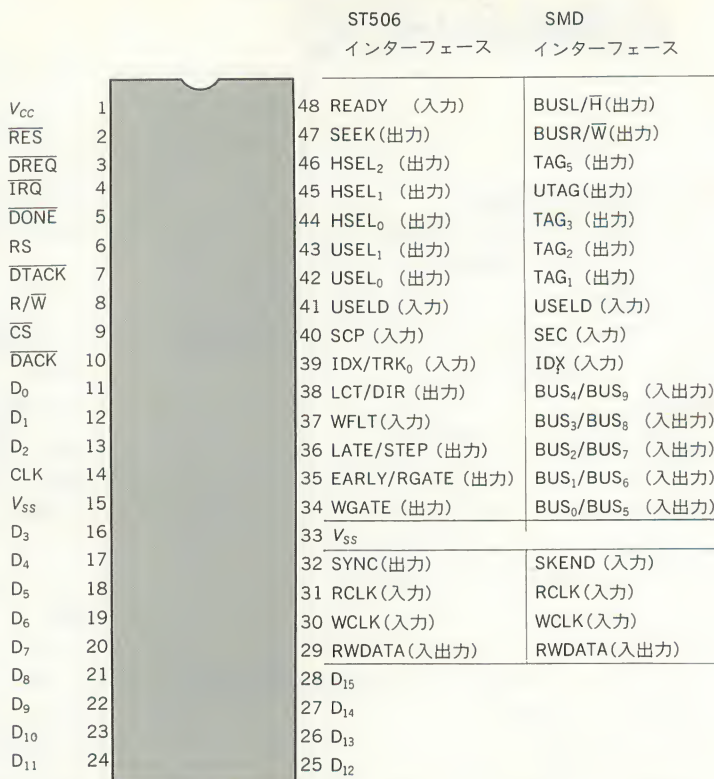
### ▶ 特 徴

- ・ 2 種類のディスク・ドライブ・インターフェース可能
- ・ ハード・セクタ (ST506 インターフェース)
- ・ ソフト・セクタ (SMD インターフェース)
- ・ データ・バス幅選択が可能 (16/8 ビット)
- ・ シリアル・データ転送レート (ハード・セクタ 20Mbps, ソフト・セクタ 10Mbps)
- ・ データ・バッファ内蔵
- ・ 256 バイト×2 面の SRAM を内蔵し、ホスト側とドライブ側の互いに独立なデータ転送の実現  
DMA なしでインターフェース可能
- ・ ディスク・データをデータ・バッファ内で自動訂正し、訂正済みのデータをホスト・システムに転送 (11 ビットまでの連続したエラー訂正が可能)
- ・ 22 種類のディスク・コマンド
- ・ CRC 発生, チェック, 32 ビット ECC 発生, チェック, 訂正
- ・ CMOS プロセス

〈図 A〉  $\mu$ PD7262 のピン配置



〈図 B〉 HD63463 のピン配置



# ● MB89342A (富士通)

MB89342A/89342AH はローカル・メモリのコントロール機能を持つ HDC です。ローカル・メモリがある場合、DMA を必要としません。ESDI のハードディスク・ドライブを制御します。

## ▶ 特 徴

### ・高速データ・レート

MB89342A	10Mbps(max, 12Mbps)
MB89342AH	15Mbps(max, 18Mbps)

### ・ローカル・メモリ制御可能(アービトレーション付)

### ・ESDI REV. B 仕様サポート

### ・ID バイト長 5 バイト/6 バイト可変

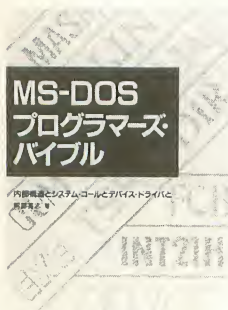
### ・8 バイト FIFO による転送タイミング制御の軽減

### ・アドレス・レジスタ構造による簡素なインターフェース

### ・オート・リトライ・モード, ノン・リトライ・モード可

### ・オート・インターリーブ・フォーマット可(ハード・セクタ)

### ・CMOS



# MS-DOS プログラマーズ・バイブル

内部構造とシステム・コールとデバイス・ドライバと...

● 阿部英志 著 ●

B 5 判/384ページ  
定価 2,500円(税込)  
送料 310円

C 言語とアセンブリ言語によるプログラム例を豊富にまじえながら、DOS の内部構造、システム・コールの使用法、デバイス・ドライバの作成法、ファイル・アクセスなどについて詳細に解説しました。MS-DOS 上のアプリケーション開発に必携の一冊です。

CQ 出版社

トランジスタ技術  
SPECIAL



## 第3章

# 記録/再生信号の変調と復調

マイクロシーケンサと訂正符号は信号記録技術の基本

松村清明

前章で述べた記録再生系に対して、NRZ 信号でインターフェースすることによりセクタ単位のリデータとしてリード/ライトする部分について説明します。

### ディスクのフォーマット

ディスク上に書かれるデータはインデックス信号(1 周ごとに 1 回出る信号で、ディスクの回転位置を示す信号)から始まり、次のインデックス信号で終わる一周の記録領域を、セクタと呼ばれる複数の小領域に分割して使用するのが一般的です。この小領域への分割方法として、セクタ・パルスといわれるパルスを発生させて一周をハードウェア的に分割するハード・セクタ方式と、記録領域のデータとしてユニークなデータを書き込むことにより、そのデータを検出してセクタの区切りとするソフト・セクタ方式の二方式があります。

この二方式の差は ID ブロック部の構成にあります。ハード・セクタ方式のセクタ・フォーマットを図 1 に、ID 部の差異を図 3～図 7 に示します。

ハード・セクタ方式におけるセクタはセクタ・パルスで始まります。ヘッド・スキヤッタから始まり、CRC、PAD までが ID 部で、このセクタのアドレスを示しています。ID 部はトラック・フォーマットなどのフォーマット時を除き、通常のライト処理時に書かれることはありません。ID 部についてはほかの方式とともに後述します。

これに対しデータ・スキヤッタから始まり ECC、PAD までがデータ部で、PLL シンクから PAD までがライト処理において書かれます。したがって書き継

ぎ点(部分的に上書きされるためにリード・パルスに連続性がなくなる点)がデータ・スキヤッタ部と ISGAP (セクタ間ギャップ)に発生し、PLL のロックが乱れます。

### ●データ部

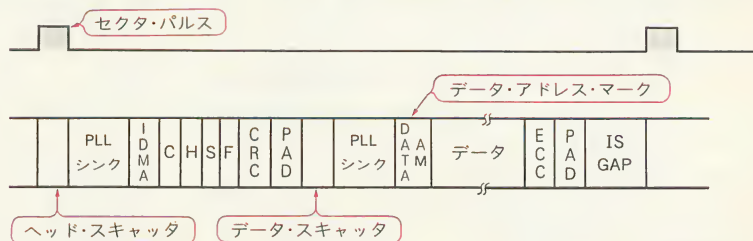
データ部を順に説明します。

データ・スキヤッタは書き込み時にそれまで ID 部をリードしていたヘッドに対し、書き込みを開始するときの時間差を吸収するためのギャップです。この時間にはリード・ライト・アンプなどのアナログ的ディレイのほかに変調/復調に必要なビットのディレイも含まれます。

この関係を図 2 に示します。ヘッドから読み出された再生波形(a)は検出回路のディレイで若干遅れて検出パルス(b)になります。復調器の中ではデコードのために 4 ビット位のデータを使用するので 4 ビット遅れて復調データ(c)になります。このデータ(シリアル・データ)は HDC(ハードディスク・コントローラ)の中でシリアル・パラレル変換されるため、最大で 8 ビット遅れます。つぎに書き込みをスタートして書き込みデータ(d)を出力しますが、この間にはディスク・コントローラの内部処理時間が必要なので少なくとも数ビット分の遅れになります。書き込みデータは変調器で変調され、変調データ(e)になります。ここで 1～2 ビット遅れ、ヘッドを通して書き込まれます。

このように変調方式が RLL の場合には 10 ビットから 20 ビット程度の遅れが発生します。したがって ID 部をリードしていた状態から直ちにデータ部を書き始めても、ID 部とデータ部の間に隙間ができてしまいます。これを吸収するために少し大きめに開ける

〈図 1〉  
ハード・セクタ方式のフォーマット



The block diagram illustrates the signal flow in a magnetic tape recording system. An input signal (f) is split into two paths. One path goes through an amplifier (a) to an equalizer (イコライザ), then to a detection circuit (検出回路), which outputs a detection pulse (b). The other path goes through an amplifier (e) to a modulator (変調), which outputs a modulated signal (d). The detection pulse (b) is fed into a PLL (Phase-Locked Loop), which outputs a demodulation signal (c) to a demodulator (復調). The demodulated signal (c) is then sent to the HDC (High Density Control) block. The timing diagram below shows the relationship between these signals and the recorded data.

(a) 再生波形 (Regeneration waveform)  
 (b) 検出パルス (Detection pulse)  
 (c) 復調データ (シリアル・データ) (Demodulated data (Serial data))  
 (d) 書き込みデータ (Serial data) (Write data (Serial data))  
 (e) 変調データ (Modulation data)  
 (f) 書き込み電流 (Write current)

PLL シンクは ID 部に対して PLL をロックするための領域で、データ部の PLL シンクと同等です。

セクタ・パルス

ID アドレス・マーク

セクタ番号

フラグ

ヘッド・スキャッタ	PLL シンク	ID AM	CYL No.	H No.	S No.	FLAG	C R C	P A D	データ・スキャッタ
-----------	---------	-------	---------	-------	-------	------	-------	-------	-----------

シリンダ番号

ヘッド番号

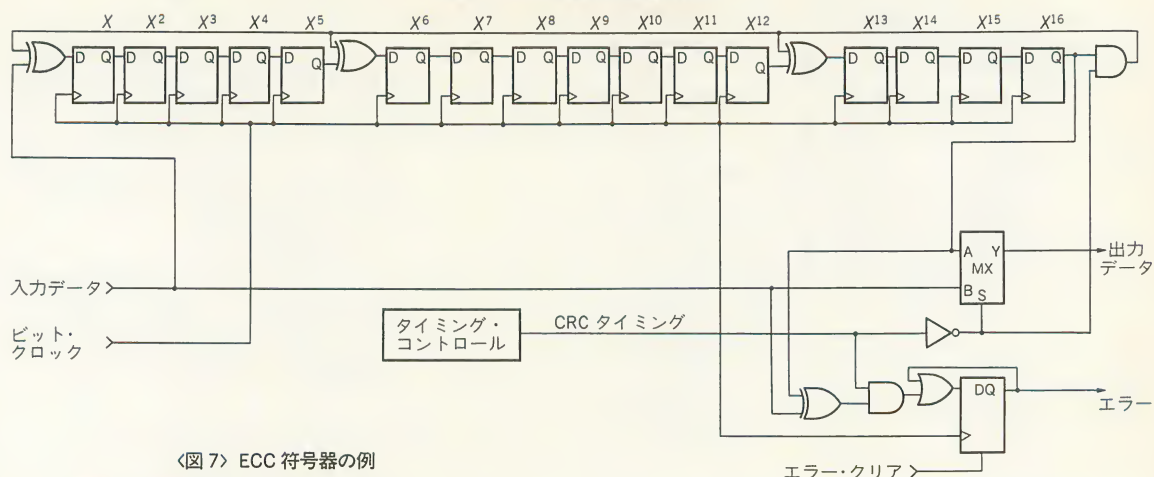
PLL シンク	ID AM	CYL No.	H No.	S No.	CRC	P A D	スキャッタ
------------	----------	------------	----------	----------	-----	-------------	-------

イレーズ・ ギャップ	PLL シンク	ID AM	CYL No.	H No.	S No.	CRC	P A D	スキャッタ
---------------	------------	----------	------------	----------	----------	-----	-------------	-------

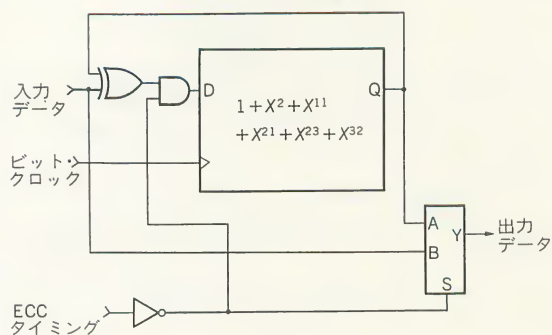
PAD はデータ部の PAD と同じ意味ですが、ID 部ののみを書くことはないのでフォーマット時にはデー



〈図 6〉 CRC 符号発生検査回路例



〈図 7〉 ECC 符号器の例



タ・スキッタと一体で処理されます。

MFМ のソフト・セクタ方式の ID ブロックの例を図 4 に示します。

ハード・セクタ方式と異なるのは ID アドレス・マークの部分です。ソフト・セクタ方式ではセクタの途中から読み始めてもセクタの先頭を見つける方法が必要です。そこで ID アドレス・マークとして **ほかのパターンと明確に区別できるユニークなパターン**をミッシング・クロックによって作っています。このような方法を使うことでアドレス・マークはどこから読み始めても検出できるわけです。

つぎに RLL の場合のソフト・セクタ方式について図 5 で説明します。

RLL は MFМ ほど冗長ではないので、ミッシング・クロックのようなユニークなパターンを作ることは困難です。そこでユニークなビット・パターンとして **24 ビットから 32 ビット程度のイレーズ・ギャップ**を作り、これを検出することでアドレス・マークとします。

これらソフト・セクタの方式はアドレス・マーク部以外はハード・セクタの場合と同様で、各方式によるデータ領域の差はほとんどありません。

## 誤り検出訂正

磁気ディスクの場合、メディア上の欠陥や記録再生系におけるジッタ、ノイズなどの原因によって必ずしも書かれた通りのデータが読まれるとはかぎりません。

したがって通常は誤り検出のためのコードを ID 部およびデータ部に付加することにより、読み出しにおいて誤りがあったかどうかを検出します。フロッピー・ディスクの場合には ID 部、データ部ともに CRC が使用されます。これに対し、ハード・ディスクの場合にはデータ部に対して **ファイア符号**と呼ばれる ECC を使用するのが一般的です。

これらの誤り検出方法はどちらも **巡回符号**と呼ばれる符号を用いるもので、**入力データ  $V(x)$  を生成多項式  $G(x)$  で割ったときの剰余**を検査ビットにします。CRC として一般に用いられる生成多項式  $G(x)$  は、

$$X^{16} + X^{12} + X^5 + 1$$

です。CRC コード発生、検査のための実際の回路を図 6 に示します。

入力データ ( $X^0$  の項) は割り算の結果  $X^{16}$  と排他的論理和を取り、それを入力します。  $X^5$  と  $X^{12}$  の項も  $X^0$  と同様です。

コードを発生する場合、データ領域では入力データがそのまま出力データとなります。CRC 領域では  $X^{16}$  のフィード・バックをマスクしてその時点で残っていた数(剰余)を出力データとします。

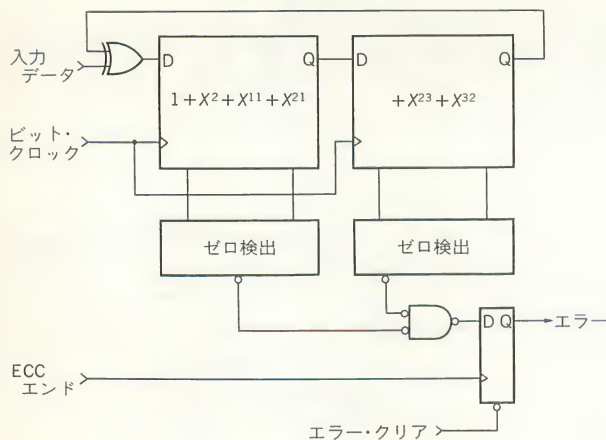
コードを検査する場合、データ領域では剰余を作るだけです。CRC 領域において入力データと剰余を比較し、異なっていればエラーとします。

つぎに ECC 符号器の例を図 7 に示します。

32 ビットの ECC の場合、生成多項式は、

$$(X^{21} + 1)(X^{11} + X^2 + 1)$$

〈図8〉 誤り検出(ECC)



です。剰余回路はこれを展開した形で実現するのが普通です。上の式を展開すると、

$$X^{32} + X^{23} + X^{21} + X^{11} + X^2 + 1$$

となります。ECCの生成の場合は多項式生成部分が異なる以外ほとんど同じで、データ領域の終わりに剰余をECCデータとして出力します。

つぎに図8で誤り検出時の動作を説明します。データ領域とそれに続くECC部の終わりまでをECC生成回路に入力し、その時点で剰余が0( $X^1 \sim X^{31}$ がすべて0)ならば誤りはありません。

もし0でなかった場合には誤りを含むことを表します。

この場合は図9に示すように入力データを0として逆方向に巡回させ、下21ビットがすべて0になるところまでシフトします。全入力ビット分シフトしても下21ビットがすべて0にならない場合には**訂正不能エラー**です。下21ビットがすべて0になったときの $11$ ビットが**シンδροーム**、シフトした数がECC領域もふくめた最終ビットからシンδροーム・データの最下位ビットまでのビット数です。ここでシンδροームとは病気の症候という意味から来た言葉で、誤りの症状を表すデータであることからこのように呼ばれます。

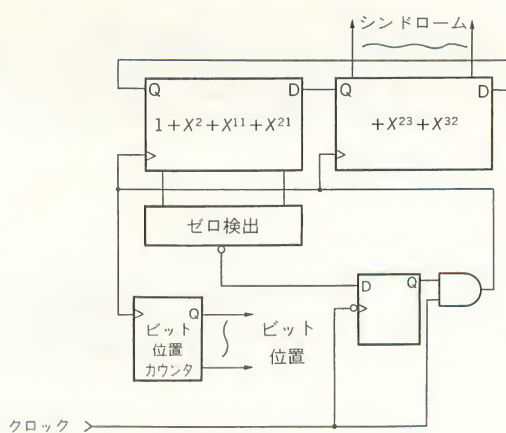
図10に得られたシンδροームおよびビット位置からデータ訂正する方法を示します。

このように**データとシンδροームの排他的論理和を取ることによってデータを訂正します**。

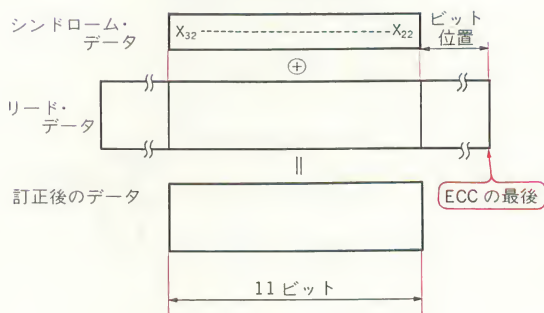
ここでは単純にシフトしていく方法でシンδροームを得る方法を述べましたが、この方法では最悪時、全ビット数分シフトする必要があります。

現在のハードディスクでは**磁気媒体に欠陥がある領域をあらかじめサーティファイして検出しておき、代替処理をして使用します**。したがって**通常の使用でECCによる訂正が発生することはまずありません**。

〈図9〉 シンδροーム生成



〈図10〉 データの訂正(ECC)



これに対して光磁気ディスクなどではエラーが発生することを前提として処理され、リアルタイム訂正が必要になるので**リード・ソロモン・コード**が使用されています。

今後は磁気記録方式においても記録密度の上昇とともにこのようなリアルタイム性のある誤り訂正能力を持ち、誤りが含まれていることを前提としたシステムが出現してくるかもしれません。

## フォーマッタの動作

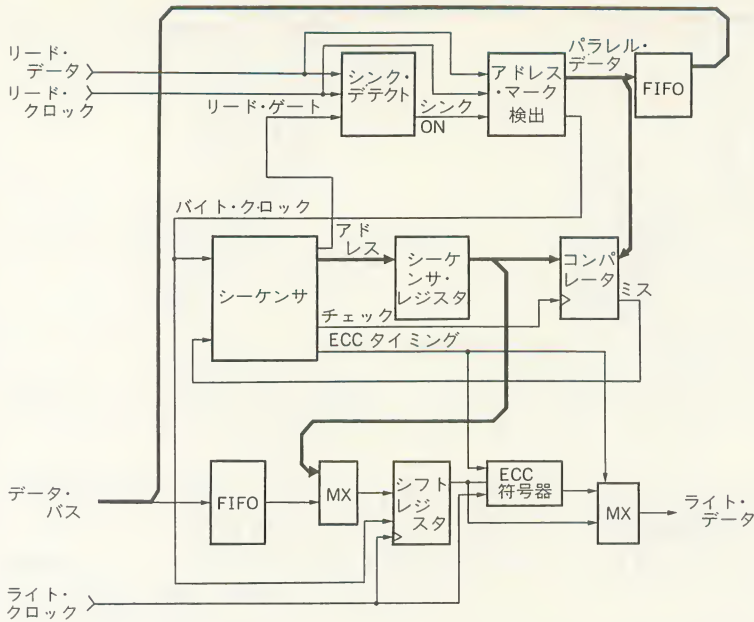
フォーマッタは通常マイクロ・プログラム・シーケンスで組まれており、シーケンス自体はマイクロ・プログラムにより実行されています。フォーマッタの概略ブロック図を図11に示します。また各部の動作を以下で順に述べていきます。

前述のようにトラックはセクタに分割されており、セクタごとに同一の処理を行うので1セクタの処理について説明を加えていきます。

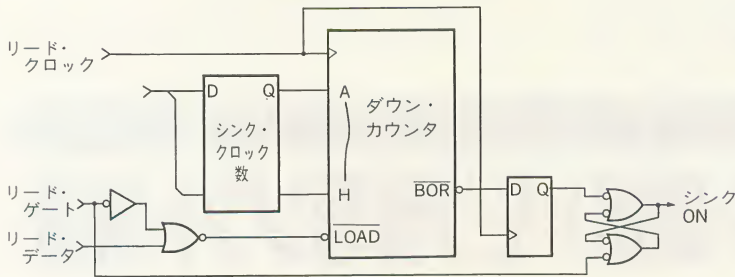
また、セクタは**ID部とデータ部**により構成されます。ライトとは目的のID部を見つけ、その後続くデータ部にデータ・ブロックを書き込むことです。目的のID部とは、**ID部の中のシリンドラ、ヘッド、セク**



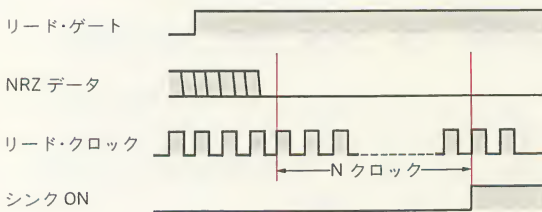
〈図 11〉 フォーマッタの概要



〈図 13〉 シンク・デテクト



〈図 14〉 シンク・デテクト・タイミング



タがすべて目的とするセクタの値と合う ID 部を指します。ライト動作のシーケンスの流れを図 12 に示します。

- (1) ID 部をリードする準備ができると、セクタ・パルスを見つけにいきます。
- (2) セクタ・パルスが見つかったらヘッド・スキップ分待ってリード・ゲートを ON し、シンク・パターンを検出しにいきます。

ヘッド・スキップとは図 2 で述べたようなリード・ライト間の総合的なディレイにより発生するライト時とリード時の時間差を指すものです。

〈図 12〉 ライト動作のシーケンス

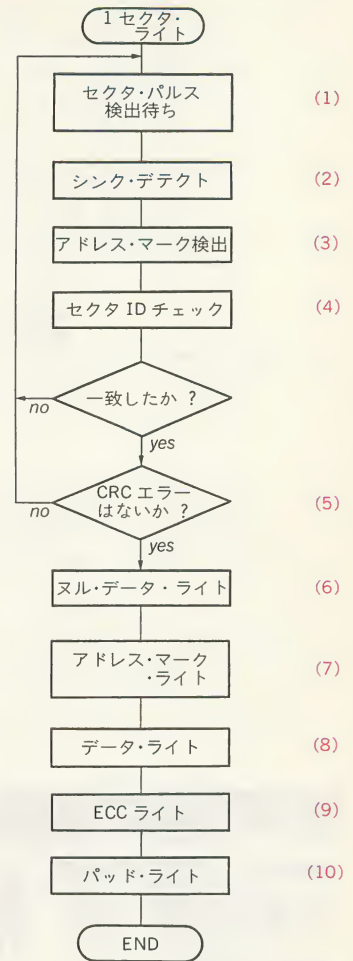


図 13 にシンク・デテクトの回路を示します。

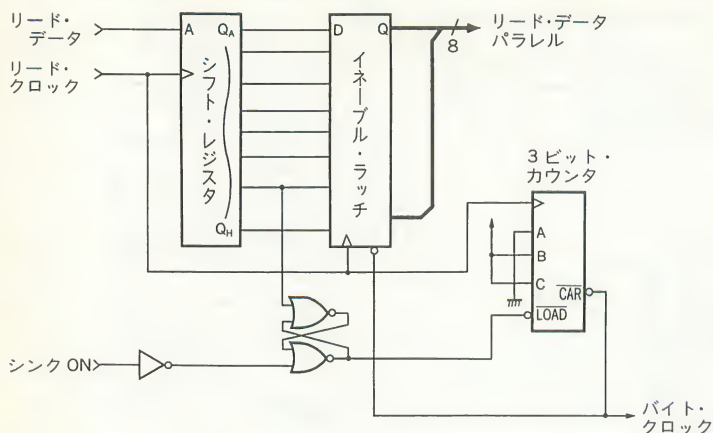
リード・データが 1 かリード・ゲートが OFF のとき、あらかじめレジスタに設定されているシンク・クロック数がダウン・カウンタにロードされます。したがってリード・ゲートが ON してから、0 のリード・データがシンク・クロック数分続くとシンク・デテクトとされ、シンク・オンがアクティブになります。これにより VFO がリード・データに同期したことを確認します。このときのタイミングを図 14 に示します。

- (3) シンク・パターンが見つかったら ID アドレス・マークを検出しにいきます。

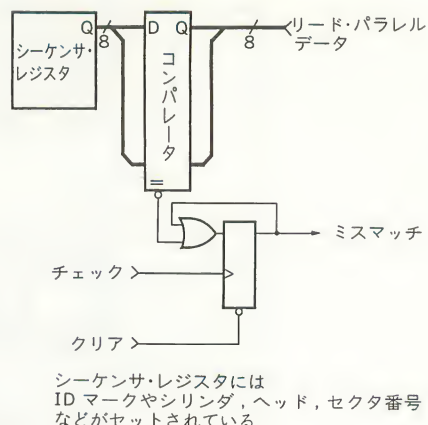
図 15 にアドレス・マーク検出回路を示します。

シンク・データの後、初めて 1 が出るのはアドレス・マークなのでこのビットがシフト・レジスタの  $Q_H$  にくるタイミングでバイト・ロック信号を作り、以後このバイト同期によってリード・データを直並列変換します。アドレス・マークが検出されたかどうかはマイクロ・プログラム・シーケンサでテストされ、エラーなら (1) へ戻ってつぎのセクタを見つけようとします。

〈図 15〉 アドレス・マーク検出(シリアル-パラレル変換)



〈図 16〉 セクタ・データ・テスト



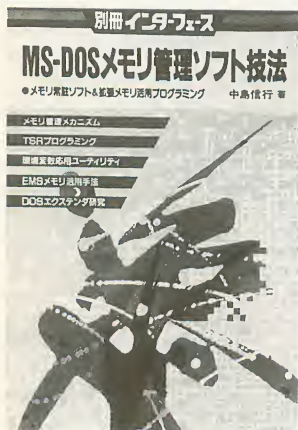
- (4) マイクロ・プログラム・シークンサはバイト・クロックごとに ID ブロック内の各データを目的とするセクタのデータと一致するかテストしていきます。
- (5) CRC ブロックをテストし、エラーであれば(1)へ戻ってつぎのセクタを見つけようとします。もし正常で、シリンダもしくはヘッドが異なっているときにはトラック・ミスマッチで異常終了です。セ

クタのみ異なっていれば(1)へ戻ってつぎのセクタを見つけようとします(図 16)。

- (6) 目的のセクタであれば、リード・ゲートを OFF し、ヘッド・スキップ時間が経過した後ヌル・データでライト・ゲートを ON します。PLL シンク長分ヌル・データを書きます。
- (7) シークンサ・レジスタの内のデータ・アドレス・マークをシフト・レジスタにロードし、アドレス・マ

## 別冊インターフェース

好評発売中



# MS-DOSメモリ管理ソフト技法

## ●メモリ常駐ソフト&拡張メモリ活用プログラミング

中島 信行 著, B5判, 2色, 224頁, 定価1,650円(税込)

MS-DOSを使いこなそうという中級プログラマの方のために、MS-DOSプログラミングのうち、メモリ管理関連手法に的を絞って、徹底的に解説しました。また、プログラミング例を多数掲載しましたが、各プログラムは、どれもMS-DOSを活用するための実用ユーティリティですので、読み進むうちに、プロのプログラミング・ノウハウが自然と身に付きます。

● 本書の内容と掲載ユーティリティ ●

### 1章 メモリ管理メカニズムとメモリ常駐プログラムの基礎

### 2章 TSRプログラミング手法を用いた各種ユーティリティ作成集

- OEMのシリアル番号検査ユーティリティ (DOS汎用)
- フォント・テーブル変更ユーティリティ (AX用)
- ハード・ディスク・ブート・ユーティリティ (98用)
- NUMキー・ロック・ユーティリティ (ハンドヘルド型98用)
- CRT監視ユーティリティ (98用)
- 子プロセス起動ユーティリティ (DOS汎用)

- TSR型デバイス・ドライバ (98/3100用)

### 3章 環境変数の活用法と応用ユーティリティ作成

- ドライブ名登録ユーティリティ (DOS汎用)
- 環境変数エディタ (DOS汎用)
- ディレクトリ・プッシュ/ポップ・ユーティリティ (DOS汎用)

### 4章 EMSメモリ活用の基礎とEMMプログラミング手法

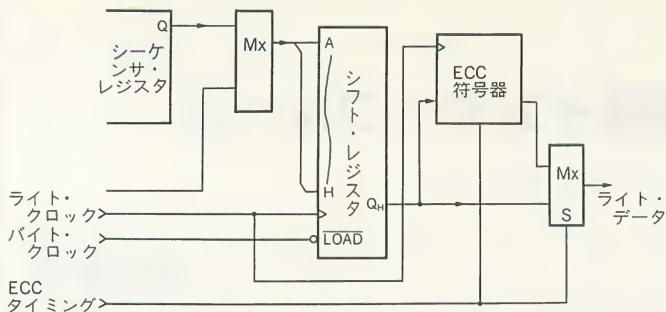
- MCBマーク/リリース・ユーティリティの作成

### 5章 80286/386用DOSエクステンダの活用研究

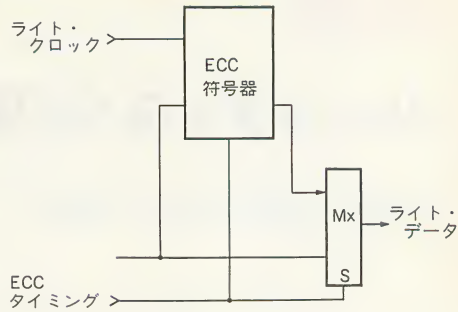
- 80386用DOSエクステンダ: 386 | TOOL BOX
- 80286用DOSエクステンダ: DOS/16M



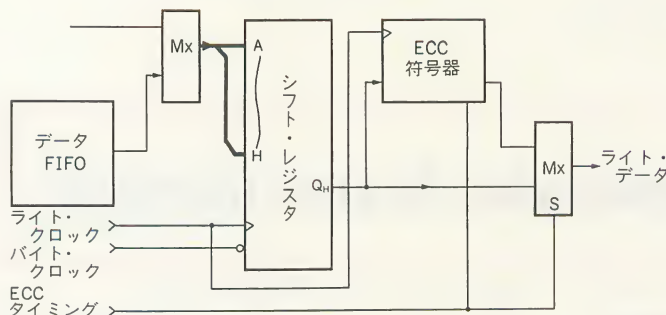
〈図17〉 データ・アドレス・マーク・ライト



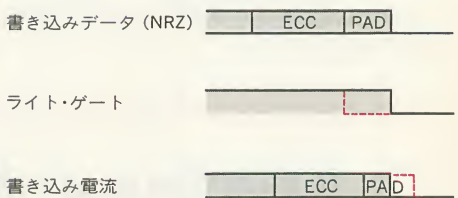
〈図19〉 ECC ライト



〈図18〉 データ・ライト



〈図20〉 パッド・データ



ークを出力します。また、ECC 符号器はこの時点から ECC を生成しはじめます(図 17)。

(8) データ FIFO の内容をシフト・レジスタを通して順次出力します。ECC 符号器はこの間 ECC を生成し続けます(図 18)。

(9) データが終了すると ECC 符号器はレジスタの内容を出力し、ECC データを書きます(図 19)。

(10) パッド・データを書き、ライト・ゲートを OFF します。

図 20 に示すように HDC が ECC データを出力した後、ただちにライト・ゲートを OFF すると NRZ データからヘッドの書き込み電流になるまでの遅れ(遅れについては図 2 を参照)によって **ECC データの途中でライトが中断されてしまいます**。そこで、ECC データが最後まで書かれるよう余分にライトされるデータを**パッド・データ**といいます。

以上でセクタのライトは終了です。リードにおいてはデータ部に対しても ID 部と同様にステップ(2)から(5)を実行します。データが FIFO を通して HDC から外へ読み出されることと CRC チェックのかわりに ECC チェックが行われることが異なるだけです。

また、ここでは DMA で HDC とメモリの間のデータがやりとりされることを前提として、データ FIFO があるものとして説明しました。しかし、HDC の内部もしくは外部に専用メモリを持つタイプの HDC の場合にはデータ FIFO がないものもあります。

ハードディスク・コントローラはこのようなフォー

マット処理を内部で実行するわけですが、**時間依存性がきわめて高い処理**であり、HDC をコントロールするファームウェアもこの時間依存を考慮してチューニングするにはかなりの困難がともないます。

また、記録方式やシーク系のコントロールも年々変化し、方法論の変化も手伝ってメガバイトあたりの単価も急速に低下しています。

ハードディスク・メーカ以外の者が HDC を直接コントロールするようなハードウェアおよびソフトウェアを作ることは技術的にもコスト的にもほとんど意味がなくなってきたといえるでしょう。

そこで、次章で述べる SCSI インターフェースを通じてハードディスクを使用することがますます重要になっていくと思われます。

#### ●参考・引用文献●

- (1)\*モハメッド・マニア他：ファイア符号用チップを用いてディスクのデータ誤りを訂正，日経エレクトロニクス，1981，8，17，pp.155-162，日経マグロウヒル。
- (2) 原島博他：誤り訂正符号化技術の応用事例，トリケップス，1986。
- (3) NEC，μPD7261 HDC ユーザーズマニュアル，1983。
- (4) NEC，μPD7263 HDC ユーザーズマニュアル，1986。
- (5) NEC，CPD データブック(μPD72061，μPD7262 他)，1989。
- (6) 日立，HD63463 ユーザーズマニュアルズ，1985。
- (7) 富士通，ハードディスクコントローラ MB89342A ユーザーズマニュアル，1989。
- (8) Adaptec, inc. AIC-270 Disk Controllers, 1986。
- (9) Western Digital, WD5010-10, Winchester Controller, 1986。

## 第4章

# ハードディスクの高速インターフェース

SASI, SCSI が今のところ標準

松村清明

パーソナル・コンピュータ用インターフェースとして一般的な SCSI とその原形となった SASI について説明します。

## ① SASI(Shugart Associates System Interface)

SASI はシュガート社がマイクロプロセッサを CPU とするような小規模コンピュータの周辺機器用インターフェースとして考案しました。これは主に磁気ディスク装置のインターフェースを目的としたものです。

当時小型ハードディスク・インターフェースはフロッピー・ライク・インターフェースといわれる ST506 インターフェースが主流でした。しかし、データ・レートがフロッピー・ディスクの 10 倍(5Mbps)であるため、インターフェースを簡単に作ることができませんでした。

そこで、タイミング的に自由度が高く 10~20 本程度の TTL で簡単にインターフェースできる SASI が作られ、マイクロコンピュータ・システムやパーソナル・コンピュータにハードディスクを接続するために使用されるようになりました。

### SASI バスのシステム構成

ホスト(SCSI ではイニシエータ)の数は 1 台に固定でアドレスはもちません。コントローラ(SCSI ではターゲット)は複数台接続することができます。

図 1 に SASI バスの構成例を示します。

セレクション・フェーズ(SCSI バスの状態遷移の項を参照)で、SASI バスのデータ・バスの  $-DB_0$  がアクティブなときに選択されるターゲットが ID=0、 $-DB_7$  がアクティブなときに選択されるターゲットが ID=7 として最大 8 台まで同一バスに接続できます。

ターミネータは両端の機器が固定的に持ち、機器の電源によりターミネータ・パワーが供給されます。SASI バスにはターミネータにバスから電力を供給する考え方はありません。

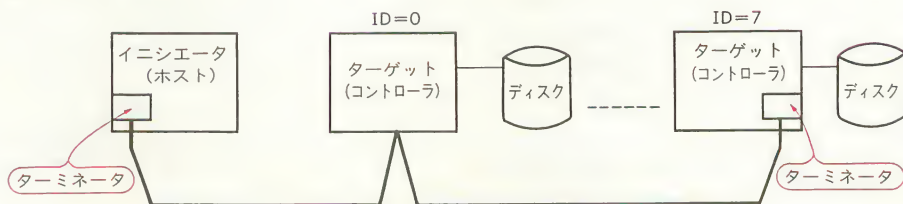
以下の説明では SCSI との表現の混同をさけるためホストとコントローラを SCSI の呼び方であるイニシエータとターゲットに統一して説明します。

SASI バス上のすべての信号は、SCSI のシングル・エンド型と同様、負論理(“L”でアクティブ)です。各信号の詳細については SASI バス信号の項を参照してください。

### SASI バスの状態遷移

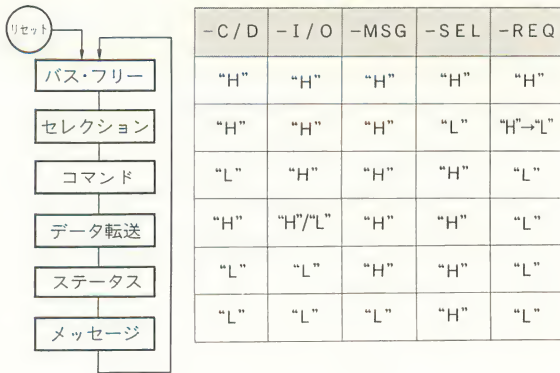
SASI バスの遷移状態は後に述べる SCSI バスと比較すると単純で、図 2 のように常にバス・フリー、セ

〈図 1〉  
SASI バスの接続例





〈図2〉 バスの状態遷移



〈表1〉 コントロール・ラインとフェーズ

信号				フェーズ名	データ転送
-SEL	-MSG	-C/D	-I/O		
"H"	"H"	"H"	"H"	データ・アウト	イニシエータからターゲットへ
"H"	"H"	"H"	"L"	データ・イン	ターゲットからイニシエータへ
"H"	"H"	"L"	"H"	コマンド	イニシエータからターゲットへ
"H"	"H"	"L"	"L"	ステータス	ターゲットからイニシエータへ
"H"	"L"	"L"	"L"	メッセージ	ターゲットからイニシエータへ
"L"	"H"	"H"	"H"	セレクション	イニシエータからターゲットへ
"H"	"H"	"H"	"H"	バス・フリー	

レクション、コマンド、データ転送(コマンドの種類により、ない場合がある)、ステータス、メッセージ、バス・フリーの各フェーズの順で動作します。

図2にSASIバスの状態遷移を示します。

現在のフェーズはコントロール・ラインによって決定されます。コントロール・ラインとフェーズの関係を表1に示します。-SELはイニシエータがドライブするライン、-MSG、-C/D、-I/Oはターゲットがドライブするラインです。いずれのラインもオープン・コレクタでドライブされる負論理の信号です。

イニシエータはひとつと決まっているのでSCSIでのアービトレーション・フェーズのようなターゲット、イニシエータ間のバス専有のためのフェーズはまったくありません(アービトレーション・フェーズについては、SCSIの項で説明)。

#### (1) バス・フリー・フェーズ

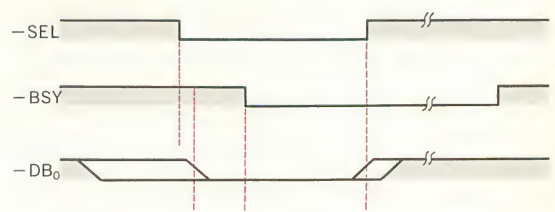
バス上のすべての機器がバスをドライブしていない状態で、-BSYが"H"(インアクティブ)です。

#### (2) セレクション・フェーズ

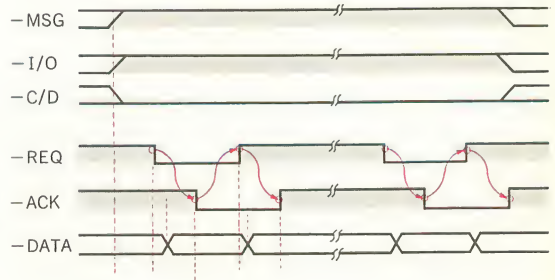
イニシエータがターゲットを選択します。

イニシエータはバス・フリー・フェーズであることを確認してデータ・ラインにターゲットのIDを出力

〈図3〉 セレクション・フェーズのタイミング



〈図4〉 コマンド・フェーズのタイミング



し、-SELをアクティブにすることによってターゲットを選択します。選択されたターゲットは-BSYをアクティブにして選択されたことを示します。このときのタイミングを図3に示します。

#### (3) コマンド・フェーズ

イニシエータからターゲットにコマンドが送出されます。-C/D="L"、-I/O="H"、-MSG="H"でコマンド送出の状態で-REQがアクティブ("L")となります。ここで-REQ、-ACKのハンドシェイクでイニシエータよりターゲットにコマンドが送られます。

コマンドの長さは6バイト固定です。このときのタイミングを図4に示します。

#### (4) データ転送フェーズ

イニシエータからターゲット、またはターゲットからイニシエータへデータが転送されます。-MSG="H"、-C/D="H"となると、データ転送モードとなります。-I/Oが"H"のときイニシエータからターゲット、"L"のときターゲットからイニシエータへデータ転送されます。このときのタイミングを図5および図6に示します。

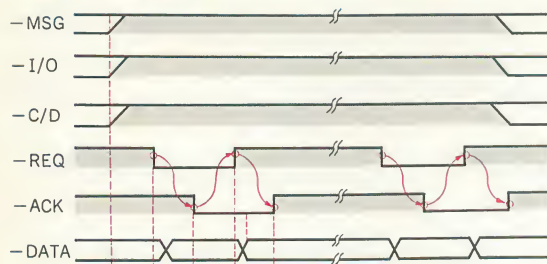
#### (5) ステータス・フェーズ

ターゲットからイニシエータへステータス・バイトを送出します。ステータス・バイトの形式を図7に示します。

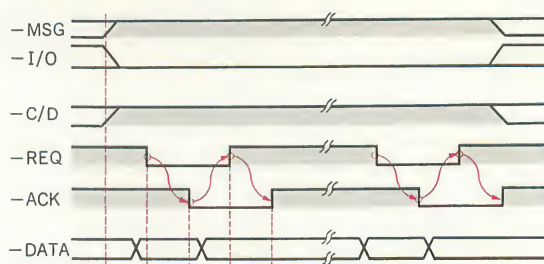
DB0がパリティ・エラー、DB1はコマンド実行中のエラー、DB2~DB4は使用しません。通常0です。DB5~DB7は論理ユニット番号(SASIコマンドの項を参照)に対応しています。

-C/D="L"、-I/O="L"、-MSG="H"で

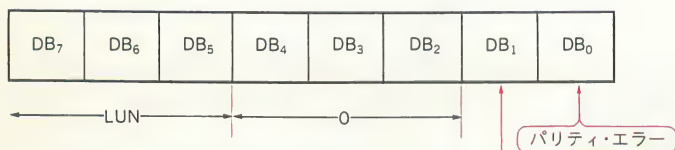
〈図5〉データ転送フェーズ(イニシエータからターゲット)



〈図6〉データ転送フェーズ(ターゲットからイニシエータ)



〈図7〉ステータス・バイト



LUN は論理ユニット番号.

実行エラーはコマンド実行におけるエラー.

パリティ・エラーはデータ転送中のパリティ・エラー.

〈図10〉コマンド・ディスクリプタ・ブロック

ビット バイト	7	6	5	4	3	2	1	0
0	コマンド・グループ			オペレーション・コード				
1	論理ユニット番号(LUN)			論理ブロック・アドレス 2(MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0 (LSB)							
4	転送長							
5	コントロール・バイト							

ステータス・バイトを送出します. このときのタイミングを図8に示します.

#### (6) メッセージ・フェーズ

ターゲットからイニシエータにメッセージ・バイトとして0(SCSIバス規格における**コマンド・コンプライート・メッセージ**に当たる)を送出します. -C/D="L", -I/O="L", -MSG="L"でメッセージ・バイトを送出します. このときのタイミングを図9に示します.

#### (7) バス・フリー

メッセージ・フェーズを終了するとターゲットは-BSYを"H"にしてバスを開放します.

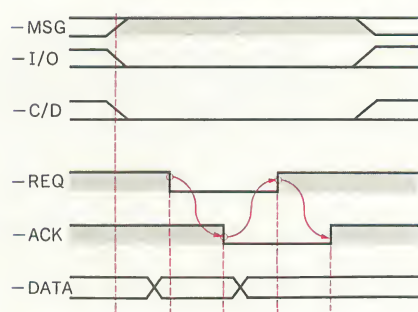
## SASI コマンド

SASI コマンドは6バイトの**コマンド・ディスクリプタ・ブロック (CDB)**で構成されます.

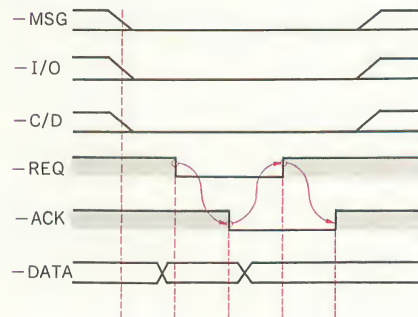
コマンド・ディスクリプタ・ブロックの一般形式を図10に示します.

0バイト目のDB<sub>7</sub>~DB<sub>5</sub>で**コマンド・グループ**を表

〈図8〉ステータス・フェーズのタイミング



〈図9〉メッセージ・フェーズのタイミング



し, 通常のディスク・コマンドはグループ0として規定されています. これに対し機器ごとの特殊コマンドはグループ7もしくは6をベンダごとに独自に使用しています.

**オペレーション・コード**はグループ内でのコマンドのナンバを示します.

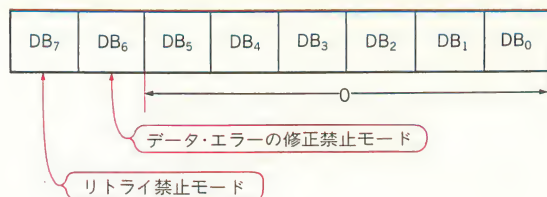
**論理ユニット番号 (LUN)**はひとつのコントローラで複数のドライブをコントロールする場合のドライブ番号です.

論理アドレス2, 1, 0は21ビットの**レコード・アドレス**で, ディスクに対してはセクタ0, ヘッド0, シリンダ0から始まる**セクタ・アドレス**です. 論理アドレスの計算は,

論理アドレス=(シリンダ・アドレス×シリンダ当たりのヘッド数+ヘッド・アドレス)×トラック当たりのセクタ数+セクタ・アドレス



〈図 11〉 コントロール・バイト



となります。

転送長はひとつのコマンドでアクセスする連続したセクタの数で、1は1セクタ、FFは255セクタを示し、0は256セクタとなります。

コントロール・バイトは図 11 のようにふたつのコントロール・ビットをもっています。DB<sub>7</sub>がリトライの禁止モードで、1の場合リトライを禁止し、0の場合リトライを許可します。DB<sub>6</sub>は修正可能データ・エラーに対してデータの修正をすることを禁止するモードです。1の場合データの修正を禁止し、0の場合データの修正を許可します。

## SASI バス信号

SASI バスは 50 ピンのコネクタにより接続され、すべての信号線は GND 線とペアで構成されています。信号はすべて負論理で、双方向のデータ・ラインと単方向のコントロール・ラインよりなります(図 12)。

以下にバス信号の意味について述べます。

### (1) -DB<sub>0</sub>~-DB<sub>7</sub>, -DB<sub>p</sub> (データ・バス)

データは-I/O が“L”でターゲットからイニシエータ、-I/O が“H”でイニシエータからターゲットの方向にドライブされます。また-SEL がアクティブなときにイニシエータがドライブし、ターゲットは 8 ビットの内の 1 ビットと-SEL の AND をとってセレクトされたことを認識します。

### (2) -BSY (ビジー)

ターゲットがドライブします。-SEL (イニシエータがドライブする)によって選択されたターゲットがドライブ(“L”)します。ターゲットがほかのコントロール・ラインをドライブするためには、まず-SELによって選択されなければなりません。このラインをドライブしている間、ターゲットはバスを専有しています。コマンドの終了によってターゲットはこのラインを解放します(すなわちバスを解放)。

### (3) -SEL (セレクト)

イニシエータがドライブします。イニシエータがターゲットを選択するために用います。選択されるとターゲットは-BSY をアクティブ(“L”)にします。イニシエータはバスがフリーである(-BSY が“H”)ことを確認し、このラインをドライブ(“L”)することで

〈図 12〉 SASI バス

ピン	信号名	イニシエータ/ターゲット
2	-DB <sub>0</sub>	↔
4	-DB <sub>1</sub>	↔
6	-DB <sub>2</sub>	↔
8	-DB <sub>3</sub>	↔
10	-DB <sub>4</sub>	↔
12	-DB <sub>5</sub>	↔
14	-DB <sub>6</sub>	↔
16	-DB <sub>7</sub>	↔
18	-DB <sub>p</sub>	↔
36	-BSY	→
38	-ACK	→
40	-RST	→
42	-MSG	→
44	-SEL	→
46	-C/D	→
48	-REQ	→
50	-I/O	→

ただし、これらのピンと対になる奇数ピンはすべて GND。また図中の矢印は信号の方向

目的のターゲットを選択できます。

### (4) -RST (リセット)

イニシエータがドライブします。

最小 400ns、最大 10s のパルスを出力することにより、バスおよびバスに接続されているターゲットを初期化するためにイニシエータがドライブします。なお、電源投入時には本信号を用いてターゲットをイニシャライズすることが望ましいとされています。

これによりバスはフリー(-BSY が“H”で、イニシエータもしくはすべてのターゲットがラインをドライブしていない状態)であり、イニシエータは任意のターゲットを選択できます。

### (5) -I/O (インプット/アウトプット)

データ・ラインのドライブ方向を決定する信号でターゲットがドライブします。

“L”レベルでターゲットがデータ・ラインをドライブし、イニシエータがデータを受け取ることを示します。

“H”レベルでイニシエータがデータ・ラインをドライブし、ターゲットがデータを受け取ることを示します。

### (6) -C/D (コマンド/データ)

ターゲットがドライブします。

データ・ライン上のデータがコマンドかデータかを示します。

“H”レベルでデータ・フェーズであることを示します。

### (7) -MSG (メッセージ)

ターゲットがドライブします。

“L”レベルでメッセージであること、すなわちコマンドの実行完了を示します。

#### (8) -REQ(リクエスト)

ターゲットがドライブします。

-ACK(イニシエータがドライブする)と対で用いられ、データのハンドシェイクを行います。-I/O, -C/D, -MSG との関係により、転送モードが規定されます。

#### (9) -ACK(アクノリッジ)

イニシエータがドライブします。

-REQ(ターゲットがドライブする)と対になり、データのハンドシェイクに用いられます。

ターゲットが-REQをドライブ("L")するとイニシエータはそれに対して-ACKをドライブ("L")し、応答することによってデータのやりとりが行えます。

### SASI コマンドの詳細

表2にSASIバスのディスク・コマンドの一部を示します。

ディスク・アクセスのコマンドとして共通で使用されていたのはこれらのグループ0のコマンドのみで、

## SASI インターフェース回路

簡単なSASIコントローラ(イニシエータ)の回路例を図Aに示します。

アービトレーション・フェーズがあるSCSIと異なり、SASIはタイミングが単純です。そこで、マイクロコンピュータと接続する場合には図のような簡単な回路でSASIディスクなどと接続することができます。

#### (1) データ・バッファ

入力時には-REQがアクティブならばデータは確定しているの、直接SASIバス上のデータをインプットし、インプットの終わりで-ACKを発生させます。

出力時には-REQがアクティブな状態でアウトプットし、ラッチしてアウトプットの終わりで-ACKを発生させます。

セレクト時にはセレクトするIDをあらかじめアウトプットしておき、-SELをアクティブにすることでターゲットをセレクトします。

#### (2) REQ/ACK ハンドシェイク

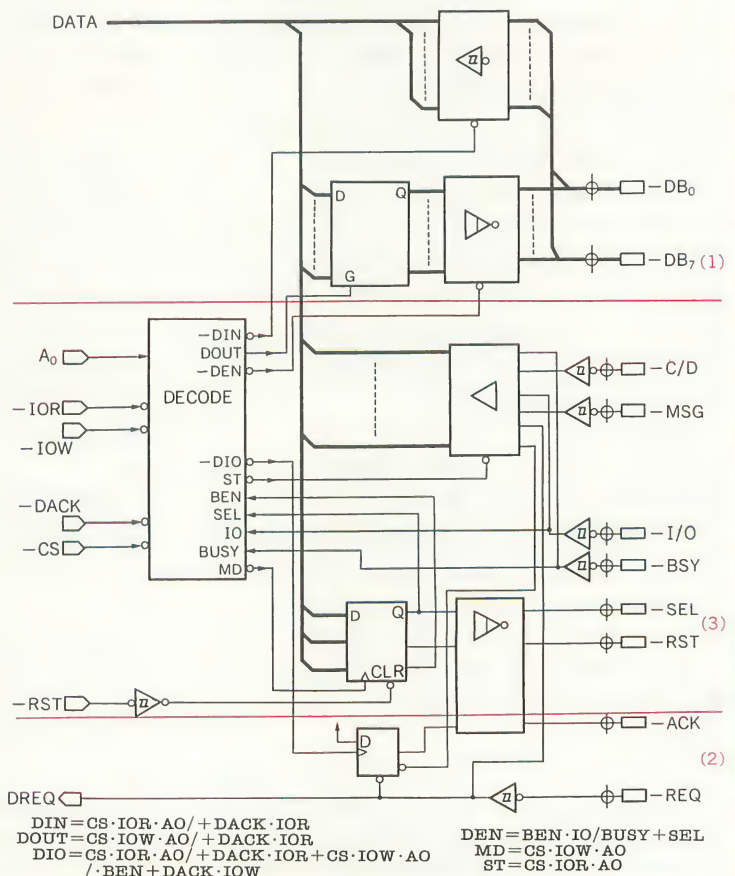
-REQがアクティブな状態でデータをインプットもしくはアウトプットすると-ACKをアクティブにし、-REQがインアクティブになるまで-ACKの出力を続けます。

#### (3) ステータス、コントロール

-SELはターゲットをセレクトするためのラインです。データ・バッファIDをあらかじめセレクトしておき、このラインをアク

ティブにします。-RSTはSASIバスのリセット・ラインです。-C/D, -MSG, -I/Oおよび-BSYはターゲットからのフェーズを表す信号で、これにもとづいてバスのドライブをコントロールしたり、プログラムによってデータの流れをコントロールします。DECODEは各ICをコントロールする信号を発生するためのPLDです。

〈図A〉 簡単なSASIインターフェース





〈表 2〉 SASI バスのディスク・コマンド

OOH	テスト・ユニット・レディ
O1H	リゼロ・ユニット
O3H	リクエスト・センス
O8H	リード
OAH	ライト

〈図 13〉 テスト・ユニット・レディ

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 14〉 リゼロ・ユニット

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 15〉 リクエスト・センス

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

それ以外に必要なコマンドはコントローラのメーカーによってグループ 6, 7 などに独自に決められていました。

グループ 0 のコマンドとしてこのほかにフォーマット・コマンドも使用されていましたが、代替処理を行わないものが多く、実際的にはコントローラによりフォーマットの方法はまちまちでした。

これらのコマンドの詳細は SCSI の項で述べます。

#### (1) テスト・ユニット・レディ (OOH)

この命令はターゲットがレディ状態であるかどうかを調べるための命令です。命令に対してターゲットはリード/ライト動作を含むすべての命令の実行が可能となるとき、**正常終了**(ステータス OOH)し、その他のときには**チェック・コンディション**(ステータス O2H)となります(図 13)。

〈図 16〉 センス・データ(4 バイト)

ビット バイト	7	6	5	4	3	2	1	0
0	センス・コード							
1	LUN			論理ブロック・アドレス 2(MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0(LSB)							

〈図 17〉 リード

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0
1	LUN			論理ブロック・アドレス 2(MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0 (LSB)							
4	転送長							
5	コントロール・バイト							

〈図 18〉 ライト

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0
1	LUN			論理ブロック・アドレス 2(MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0 (LSB)							
4	転送長							
5	コントロール・バイト							

#### (2) リゼロ・ユニット (O1H)

この命令を受けたときには、装置内部のヘッド位置決め系の再初期化を行い、シリンダ 0 へシークします(図 14)。

#### (3) リクエスト・センス (O3H)

この命令はターゲットが**チェック・コンディション・ステータス**を送出した際にイニシエータが詳細情報を読み取るための命令です(図 15)。

ターゲットは 4 バイトのセンス・データ(図 16)を送出します。SCSI コマンドの**非拡張センス・データ**がこれと同じです。詳しくは SCSI コマンドの**リクエスト・センス**の項で述べます。

#### (4) リード (O8H)

この命令はイニシエータがターゲットに対して磁気ディスク媒体に記録されているデータの転送を要求する命令です。**コマンド・ディスクリプタ・ブロック**の論理ブロック・アドレスはデータ転送を要求する先頭のブロックを示し、転送長は転送を要求するブロック数を示します(図 17)。

#### (5) ライト (OAH)

イニシエータがターゲットに対して磁気ディスク媒体にデータの書き込みを要求する命令です(図 18)。

コマンド・ディスクリプタ・ブロックの詳細はリード・コマンドと同じです。

## ② SCSI (Small Computer System Interface)

SCSI は SASI を基本に、X3T9.2 として 1982 年 4 月からアメリカ規格協会 (ANSI) で検討が始められました。その後 2 カ月間隔で ANSI の X3 委員会により公式会議が開かれ、1986 年 6 月に ANSI により承認されアメリカ国家規格 ANSI X3.131-1986 “Small Computer System Interface” として制定されました。

SASI バスではイニシエータはひとつであったのに対し SCSI では複数のイニシエータが 1 本の SCSI バス上にあることを認めたためアービトレーション (調停) が必要になり、イニシエータもアドレスをもつことになりました。SASI では一度コネクトが確立されるとコマンド終了までバスを占有していました。これに対して SCSI では複数のイニシエータがバスを使用し、マルチプロセス OS においては同時に複数のターゲットをアクセスするためバスがフリーになるのを待つこととなるような場合が起こります。

そこでコマンド起動後データ転送がないときには一度ディスコネクトしてバスを開放し、再度リセクションすることによりバスの占有時間を短縮し、バス使用効率を上げることを可能にしました (このような例を SCSI バスの状態遷移の項で述べる)。ただし、同時にひとつのアクセスしか発生しない場合にはディスコネクトしないほうが高いスループットを得ることができます。

SASI ではコネクトが確定するとバス・リセットを除いてディスコネクトまでターゲットがすべてのシーケンスをコントロールしていました。SCSI では -ATN (アテンション) によりメッセージ・システムを通してイニシエータからターゲットに指示ができるようになりました。

### SCSI バスのシステム構成

SCSI バスには 8 台の装置が接続でき、それぞれが 0~7 の ID 番号をもちます。ID=7 の優先順位が最も高く、ID=0 の優先順位が最低です。それぞれの

役割も SASI のように固定的でなく、イニシエータ (命令を出す側) にもターゲット (命令を実行する側) にもなることができます。ただしこれはハードウェアの機能によります。

図 19 に SASI バスの接続例に相当する SCSI バスの接続例を示します。これはイニシエータ 1 台に対して 1~7 台のターゲットが接続されている例です。

ターゲットは SASI と同様に ID=0 から順に ID を割り当てられているのが一般的です。またイニシエータは逆に ID=7 から順に ID を割り当てることが多いようです。ただし重複しない任意の ID を任意の順で用いてもかまいません。

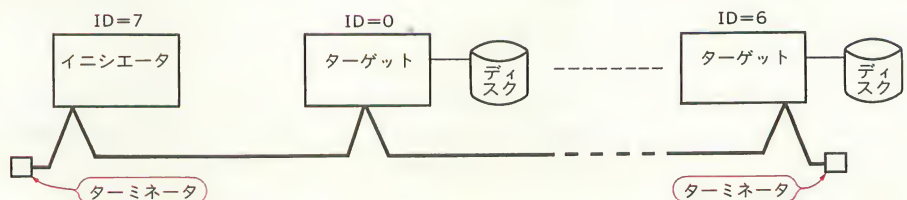
ターミネータは両端機器の外部にもち、バスの TERMPWR によりターミネータ・パワーが供給されます。各信号の詳細については SCSI バス信号の項を参照してください。また SASI バスと同様に負論理 (“L” がアクティブ) のシングル・エンド型を例として以下で説明します。

### SCSI バス状態遷移

SCSI バスのもっとも単純な状態遷移は図 2 で示した SASI の状態遷移と同じであり、SASI バスに対して上位方向の互換性をもっています。これに対してアービトレーションを行う、より一般的な SCSI バスの状態遷移を図 20 に示します。また ANSI では状態遷移中の時間を厳密に規定しています。この時間規定を表 3 に示します。以下、各状態の遷移について順に説明していきます。

- (1) いかなるフェーズにいてもリセットによってバス・フリー状態になります。またバスを占有しているターゲットはコマンド、データ、ステータス、メッセージの各フェーズからハンドシェイクの単位ごとの任意のタイミングでディスコネクトしてバス・フリー・フェーズへ戻ることができます。
- (2) バスを使用したいイニシエータもしくはターゲットはバス・フリーであることを認識 (-BSY=

〈図 19〉  
SCSI バスの接続例





“H” )してアービトレーション・フェーズに入ります。

図 21 でアービトレーションからセレクションまでの状態遷移を説明します。

SCSI デバイス (ID<sub>5</sub>) は、バス・フリー・フェーズとなるのを待ち、-BSY 信号がアクティブであることを検出後バス・フリー時間を待って SCSI バス・フリー状態終了後からバス・セット時間以内に-DB<sub>5</sub>信号とその-BSY 信号をアクティブ(“L”)にします(図 21 ではバス・セット時間は 0 に等しい)。

つぎにアービトレーション・ディレイだけほかの DB 信号を監視します。

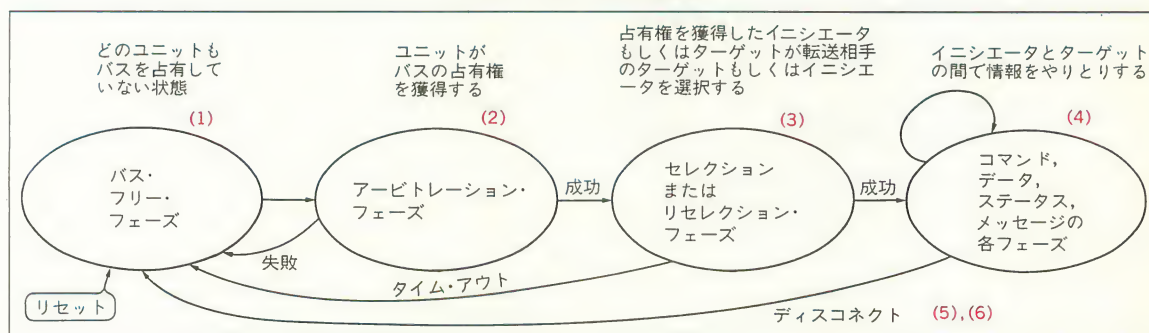
SCSI デバイス (ID<sub>7</sub>) は同時刻 (ID<sub>5</sub>より若干遅れた

時間)にバス・フリー・フェーズを検出します。ID<sub>7</sub>のデバイスはバス・フリー時間、バス・セット時間を ID<sub>5</sub>のデバイスと同じように待ったのち、-DB<sub>7</sub>信号とその-BSY をアクティブ(“L”)にします。つぎにアービトレーション・ディレイだけ ID<sub>5</sub>のデバイスと同じようにほかの DB 信号を監視します。

ここで ID<sub>7</sub>のデバイスが ID<sub>5</sub>のデバイスより優先となるので ID<sub>5</sub>のデバイスは-DB<sub>7</sub>信号がアクティブ(“L”)となっていることを検出すると-BSY と-DB<sub>5</sub>のドライブをバス・クリア・ディレイ以内に中止します。

ID<sub>7</sub>のデバイスは引き続きアービトレーション・ディレイだけ待ったのち-SEL 信号をドライブします。

〈図 20〉 SCSI バスの状態遷移



〈表 3〉 SCSI バスの時間規定

(1) アービトレーション・ディレイ (最小 2.2μs)

アービトレーション・フェーズで SCSI 装置が-BSY をアクティブ(“L”)にし ID をデータ・バスに送ってから優先順位が確定するまでの最小時間

(2) アサーション時間 (最小 90ns)

同期データ転送において-REQ もしくは-ACK の最小パルス幅

(3) バス・クリア・ディレイ (最大 800ns)

次の三つの要因が発生後、SCSI 装置が信号のドライブを終了するまでの最大時間

- バス・フリー・フェーズの検出の場合
- アービトレーション・フェーズ中、他のデバイスから SEL 信号を受けた(アービトレーション・フェイル)
- RST 信号がアクティブ(“L”)になった

(4) バス・フリー・ディレイ (最小 800ns)

アービトレーション・フェーズを開始するとき、バス・フリー・フェーズ検出から-BSY と ID をドライブするまでの最小待ち時間

(5) バス・セット・ディレイ (最大 1.8μs)

アービトレーション・フェーズを始めるとき、バス・フリー・フェーズ検出から-BSY と ID をドライブするまでの最大許容時間

(6) バス・セトル・ディレイ (最小 400ns)

信号状態変化後、バスの安定を待つべき最小時間

(7) ケーブル・スキュー・ディレイ (最大 10ns)

任意の二つのバス信号線の間の伝搬時間の最大差

(8) データ・リリース・ディレイ (最大 400ns)

-I/O 信号が“H”から“L”に変化後、イニシエータがデータ・バスのドライブを終了するまでの最大時間

(9) デスキュー・ディレイ (最小 45ns)

バス信号間の遅れの差を補正するために必要な最小時間

(10) ホールド・タイム (最小 45ns)

同期データ転送で、イニシエータまたはターゲットが-ACK または-REQ をドライブしてからデータ・バス上のデータを保持しておくべき最小時間

(11) ネゲーション時間 (最小 90ns)

同期データ転送で、REQ 信号から次の REQ 信号または ACK 信号から次の ACK 信号までのインアクティブ状態の最小時間

(12) リセット・ホールド・タイム (最小 25μs)

-RST 信号をアクティブ状態に保持しておく最小時間

(13) セレクション・アボート・タイム (最大 200μs)

セレクション、リセレクション・フェーズで SCSI 装置が選択されたことを認識後、-BSY で応答するまでの最大許容時間

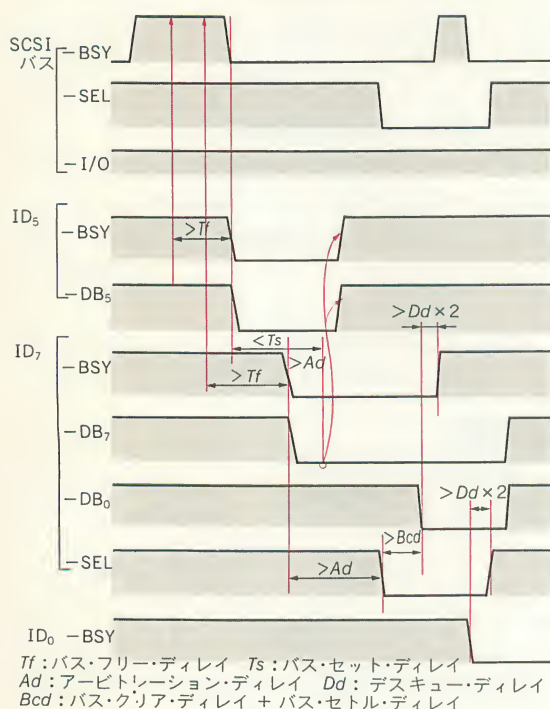
(14) セレクション・タイムアウト・ディレイ (最小 250ms 推奨)

セレクション、リセレクション・フェーズでタイムアウト処理を始めるまでに SCSI 装置からの BSY 信号を待つべき最小時間。実際にはこれよりかなり長い時間(たとえば数秒)待つことが多い

(15) トランスファ時間 (180ns～1020ns)

同期データ転送中での REQ 信号、ACK 信号の最小繰り返し時間・メッセージ・フェーズによりイニシエータ・ターゲット間で個別に規定された時間(メッセージ・システムの項を参照)

〈図 21〉 SCSI バスの状態遷移  
(アービトレーション～セレクション・フェーズ)



これで ID<sub>7</sub> のデバイスがバス使用権を獲得したことになります。図 21 の前半がこれに当たります。

(3) イニシエータならセレクション・フェーズ、またターゲットならリセレクション・フェーズに入ります。

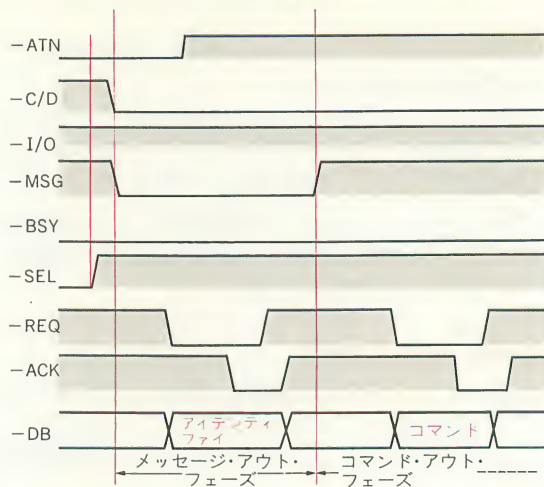
セレクション・フェーズは図 21 の後半、リセレクション・フェーズは図 23 の最後で示しています。セレクション・フェーズとリセレクション・フェーズはセレクトされた時点で -I/O が“H”か“L”かで決まり、セレクションでは“H”，リセレクションでは“L”です。

ここでは図 21 にそって ID<sub>7</sub> がイニシエータで ID<sub>0</sub> のターゲットをセレクトする場合について説明します。-SEL をドライブした後、バス・クリア・ディレイ + バス・セトル・ディレイ 待ってセレクトしたいデバイスの ID (この場合 ID<sub>0</sub>) に対応する -DB<sub>0</sub> をドライブします。

つぎに、デスキュー・ディレイの二倍待って -BSY のドライブをやめ、ターゲットが -BSY をドライブするのを待ちます。ターゲットの応答がなければタイム・アウトとしてデータ・バスのドライブをやめ、つぎに -SEL のドライブをやめてバス・フリー・フェーズへ戻ります。

ターゲットからの応答があればデス・キュー・ディレイの二倍待って -SEL のドライブとデータ・バスのドライブを終了し、つぎのフェーズへ移行します。この

〈図 22〉 SCSI バスの状態遷移(コマンド・フェーズ)



とき、アイデンティファイ・メッセージなどのメッセージを出力したい場合には -SEL のドライブをやめるより前に -ATN をドライブしておきます。

(4) セレクション・フェーズのつぎは -ATN がアクティブならメッセージ・アウト・フェーズ、-ATN がアクティブでなければコマンド・アウト・フェーズです。

図 22 にメッセージ・アウト・フェーズ(アイデンティファイ・メッセージ)を含む場合のコマンド・アウト・フェーズの開始部分のタイミングを示します。アイデンティファイ・メッセージについてはメッセージ・システムの項を参照してください。

ターゲットはセレクション・フェーズの終わりで -ATN がアクティブである間はメッセージ・アウト・フェーズと必要ならその応答のフェーズを続けます。

ここでイニシエータはアイデンティファイ・メッセージなど(詳しくはメッセージ・システムの項を参照)のメッセージを出力します。

次にコマンド・フェーズに入り、6 バイトから 12 バイトのコマンドをイニシエータがターゲットに出力します。コマンドのバイト数と内容は SCSI コマンドの項を参照してください。

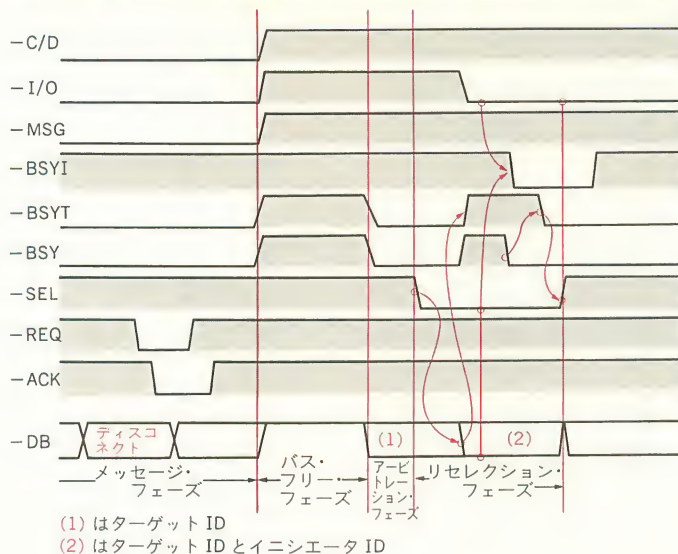
(5) アイデンティファイ・メッセージでディスコネクトできることをイニシエータがターゲットに示した場合、ディスコネクトできるターゲットはコマンドの終了までの間、もしくはデータ転送の準備ができるまでの間、ディスコネクトしてバスを開放します。

図 23 にバスの開放とその後のターゲットによるイニシエータのリセレクション・フェーズを示します。

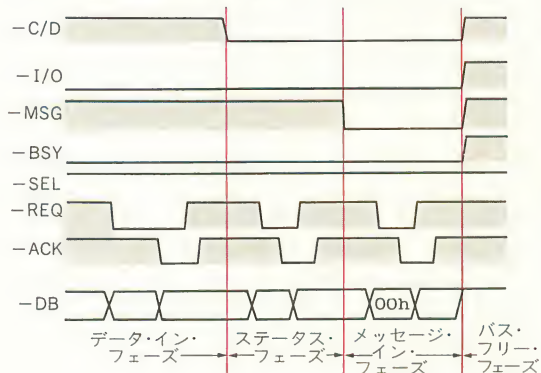
通常ターゲットはディスコネクトする前にセーブ・データ・ポインタなどとともにディスコネクト・メッセージを出力し、つぎにディスコネクトします(メッセ



〈図 23〉 SCSI バスの状態遷移(ディスコネクト, リセクション・フェーズ)



〈図 24〉 SCSI バスの状態遷移(データ転送, 終了フェーズ)



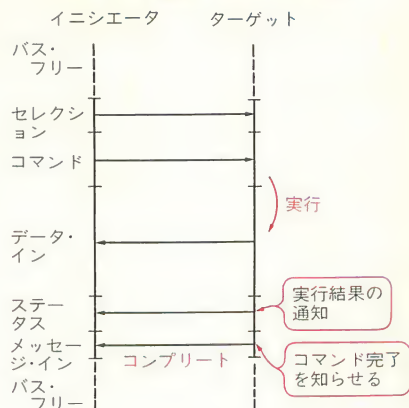
ージ・システムの項を参照)。データ転送やコマンド終了の準備が整うとターゲットはイニシエータをリセレクトします。ターゲットはアービトレーション・フェーズの後に-SEL をアクティブにし、イニシエータの ID を出力するとともに-I/O をアクティブにしてこれがリセクションであることを示します。

つぎに一度-BSY のドライブをやめ、イニシエータが応答して-BSY をドライブしてくるのを待ちます。イニシエータが応答してくるとターゲットはふたたび-BSY をドライブして-SEL のドライブを終了し、つぎのフェーズへ移行します。この間-I/O はドライブしたままです。

(6) 最後にコマンド終了とそれとともうバス・フリー・フェーズへの移行を図 24 に示します。

この例ではディスク・リードやセンス・ステータスなどのデータ・イン・フェーズに続けて終了フェーズへ移行する場合を示しています。データ転送フェーズが終了すると SASI バスの状態遷移と同様にステータス・

〈図 25〉 単純な SCSI シーケンス



フェーズに入ります。ここでターゲットは 0(正常終了)もしくは 2(エラー, 例外状態, 異常状態での終了)などのステータスをイニシエータに出力し、つぎにメッセージ・アウト・フェーズで 0(コマンド終了)を出力したのちディスコネクトします。

以上でコマンド実行にともなう一般的な状態遷移を説明しましたが、一台のディスクをアクセスするだけの単純なシステムでは SASI の項で説明したもっとも単純な状態遷移の場合にスルー・プットが最大になることに注意しなければいけません。

不必要に複雑なシーケンスにするとスループットはどんどん低下します。

### ● 単純なシーケンス

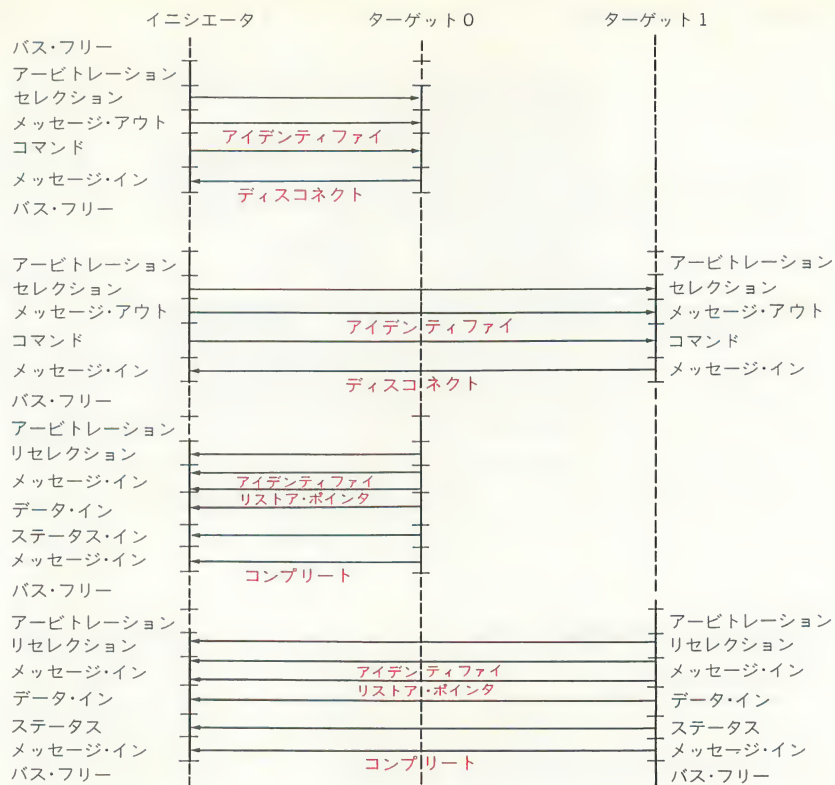
図 25 にもっとも単純な SCSI シーケンスを示します。

イニシエータはバス・フリー・フェーズであることを確認してセレクション・フェーズに入ります。セレクション・フェーズでターゲットを選んだイニシエータは、つぎにコマンド・フェーズに移行し、ターゲットはイニシエータからコマンドを得て実行します。データ・イン・フェーズではコマンド実行にともなうデータ転送を行います。つぎにターゲットはステータス・フェーズによってコマンドの実行結果をイニシエータに通知します。最後にターゲットはメッセージ・イン・フェーズでコマンド完了を通知し、バス・フリー・フェーズに移行します。

図 26 にディスコネクトのある少し複雑な SCSI シーケンスを示します。この中で使用されるメッセージについてはメッセージ・システムの項を参照してください。

バス・フリー・フェーズを確認したイニシエータはア

〈図 26〉  
ディスコネクトのある  
SCSI シーケンス



ービトレーション・フェーズによって SCSI バスを占有します。セレクション・フェーズでイニシエータはターゲット 0 を選び、-ATN をアクティブにします。メッセージ・アウト・フェーズでイニシエータはアイデンティファイ・メッセージによってディスコネクトが可能であることをターゲット 0 に通知します。コマンド・フェーズにおいてターゲット 0 はイニシエータからコマンドを得て実行します。

ここでデータのやりとりがなく、しかも実行時間の長いコマンド(フォーマットなど)やデータ転送までに時間のかかるコマンド(リードなど)においては、SCSI バスのむだな占有を防ぐためにターゲット 0 はメッセージ・イン・フェーズに入ってディスコネクト・メッセージを通知し、バス・フリー・フェーズに移行します(一時的に接続を中断する)。

この間にマルチプロセスで動いているようなイニシエータが他のターゲットに対するアクセスが必要な場合にはアービトレーション・フェーズによって SCSI バスを占有します。

セレクション・フェーズでイニシエータはターゲット 1 を選び、-ATN をアクティブにします。メッセージ・アウト・フェーズでイニシエータはアイデンティファイ・メッセージによってディスコネクトが可能であることをターゲットに通知し、コマンド・フェーズでターゲット 1 はイニシエータからコマンドを得て実

行します。

もし、ここでも実行時間がかかる場合にはメッセージ・イン・フェーズに入り、ディスコネクト・メッセージがターゲットから通知され、バス・フリー・フェーズに移行します。

データ転送の準備ができたターゲット 0 はここでアービトレーション・フェーズによってバスを占有し、リセレクション・フェーズにおいてディスコネクトしたイニシエータを選択します。ターゲットはメッセージ・イン・フェーズでアイデンティファイ・メッセージを送り、このリセレクションに対応する論理ユニットを通知し、つぎにリストア・ポインタを発行します。

ターゲットはデータ・イン・フェーズでコマンドに対するデータをイニシエータに送ります。ステータス・フェーズではターゲットからイニシエータへコマンドに対する実行結果を送ります。メッセージ・イン・フェーズではターゲットからイニシエータにコマンド完了を通知します。

ターゲット 1 もこの後ターゲット 0 と同様にリセレクションし、データ転送を行います。

このように単純な SCSI シーケンスと比べると複数のメッセージ・システムを使用するので、**イニシエータ-ターゲットのオーバ・ヘッドは大きくなりますが、バスは有効に使用されている**といえます。

それでは、それぞれフェーズについて説明します。



〈図 27〉 6 バイト・コマンド・ディスクリプタ・ブロック

ビット バイト	7	6	5	4	3	2	1	0
0	コマンド・コード							
1	LUN			論理ブロック・アドレス 2(MSB)				
2	論理ブロック・アドレス 1							
3	論理ブロック・アドレス 0							
4	転送長							
5	コントロール・バイト							

〈図 28〉 10 バイト・コマンド・ディスクリプタ・ブロック

バイト	7	6	5	4	3	2	1	0
0	コマンド・コード							
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス 3 (MSB)							
3	論理ブロック・アドレス 2							
4	論理ブロック・アドレス 1							
5	論理ブロック・アドレス 0 (LSB)							
6	0	0	0	0	0	0	0	0
7	転送長 (MSB)							
8	転送長 (LSB)							
9	コントロール・バイト							

## ● コマンド

SCSI コマンドの概要を示します。あくまでも概要を示すのみですから内容については SCSI コマンドの詳細の項を参照してください。

図 27～図 29 に 6 バイト、10 バイト、12 バイト、それぞれの **コマンド・ディスクリプタ・ブロック (CDB)** を示します。

イニシエータからのコマンドは **—MSG**、**—I/O** 信号がインアクティブ (“H”) で、**—C/D** がアクティブ (“L”) な状態でターゲットからの要求で REQ/ACK プロトコルによって、イニシエータから転送されます。コマンド・ディスクリプタ・ブロック (CDB) の先頭バイトには **コマンド・コード** が示され、**コマンド・コードのグループによって CDB の長さが決まります**。以降のバイトには **論理ユニット番号**、**先頭ブロック・アドレス**、**転送要求ブロック数**および**コントロール・バイト**が続きます。

## ● ステータス

ステータス・バイトはそれぞれのコマンド実行終了時のステータス・フェーズにおいてターゲットからイニシエータへ転送されます。図 30 にステータス・バイトの形式を示し、以下に各ステータスの詳細を示します。

### (1) 正常終了 (OOH)

本ステータスはコマンドの実行が**正常終了**したことを示します。

### (2) チェック・コンディション (O2H)

〈図 29〉 12 バイト・コマンド・ディスクリプタ・ブロック

バイト	7	6	5	4	3	2	1	0
0	コマンド・コード							
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス 3 (MSB)							
3	論理ブロック・アドレス 2							
4	論理ブロック・アドレス 1							
5	論理ブロック・アドレス 0 (LSB)							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	転送長 (MSB)							
10	転送長 (LSB)							
11	コントロール・バイト							

〈図 30〉 ステータス・バイト

バイト	7	6	5	4	3	2	1	0
0	LUN			ステータス・バイト・コード				0

センス・データがセットされる**エラー**、**例外状態**、**異常状態が発生したことを示すステータス**です。このステータスがセットされた場合、イニシエータは**リクエスト・センス・コマンド**を発行して詳細情報を受け取る必要があります。

### (3) ビジー (O8H)

イニシエータから受け取った命令に対してターゲットがビジーでその**命令の実行ができないことを示します**。イニシエータは命令を再度実行する必要があります。

### (4) リザーベーション・コンフリクト (18H)

ターゲットがほかのイニシエータによってリザーブされているため**コマンドの実行が不可能なことを示します**。

## ● メッセージ・システム

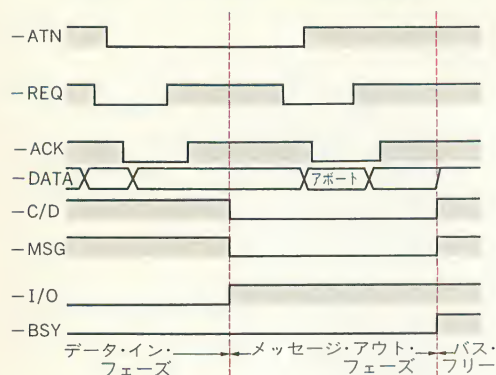
メッセージ・システムはイニシエータとターゲット間でモードをやり取りするために用いられます。

ターゲットはコマンド・コンプリート以外に数種のメッセージをサポートしています。コマンド・コンプリート以外のメッセージをサポートするイニシエータはセクション・フェーズにおいて **—SEL** のドライブをやめる前に **—ATN** をドライブ (“L”) することで **アテンション・コンディション** をつくり、**転送すべきメッセージがあることを知らせることができます**。

この操作によってターゲットは、メッセージ・アウト・フェーズに入ります。

アービトレーション機能をもつシステムのイニシエータはアイデンティファイ・メッセージを送出してディスクコネクタ/リコネクタ機能をもつことをターゲットに知らせることができます。この場合の状態遷移は

〈図 31〉 -ATN によるメッセージの起動例



SCSI バス状態遷移の項の(4)で図示されています。

またコネクト中(コネクト中のフェーズの遷移はすべてターゲットがコントロールする)にイニシエータがターゲットに対してコントロールを要求する場合、-ATN をドライブ(“L”)します。この要求に対してターゲットは**転送の切れ目でメッセージ・アウト・フェーズ**に入ります。ここでイニシエータはターゲットに後述するアボート・メッセージのようなメッセージを送ることで処理を中断できます。この状態の例を図 31 に示します。

この例はデータ・イン・フェーズにおいてイニシエータが中断した例です。

データ・イン・フェーズにおいて -ATN をアクティブにしたのでターゲットはデータの切れ目でメッセージ・アウト・フェーズに移行しました。

ここでイニシエータは**ターゲットがメッセージ・アウト・フェーズに入るまでターゲットの要求する転送を続けることが重要**です。

メッセージ・アウト・フェーズでアボート・メッセージを受け取ったターゲットはバス・フリー・フェーズに移行します。

表 4 に一般的に使用されるメッセージ・コードを示します。

#### (1) コマンド・コンプリート (00h)

ターゲットはコマンドの実行終了時にコマンド・コンプリート・メッセージを出力し、ディスコネクトしてバス・フリー・フェーズに戻ります。

#### (2) セーブ・データ・ポインタ (02h)

このメッセージはディスコネクト時にイニシエータに対して**転送中のデータのポインタをセーブ**することを要求するメッセージです。

実行中のコマンドのデータが全然転送されていない場合にはディスコネクト時にディスコネクト・メッセージのみを出力し、一部のデータが転送済みの場合にはセーブ・データ・ポインタ、ディスコネクト・メッセージを出力してバス・フリー・フェーズにするのが普通

〈表 4〉 メッセージ・コード

16 進コード	ディスクリプション	メッセージ IN/OUT
00h	コマンド・コンプリート	IN
01h	拡張メッセージ	IN / OUT
02h	セーブ・データ・ポインタ	IN
04h	ディスコネクト	IN
05h	イニシエータ・ディテクト・エラー	OUT
06h	アボート	OUT
07h	メッセージ・リジェクト	IN / OUT
08h	ノー・オペレーション	OUT
09h	メッセージ・パリティ・エラー	OUT
0Ch	バス・デバイス・リセット	OUT
80h, C0h	アイデンティファイ	IN / OUT

です。イニシエータはリセレクト時にターゲットから**アイデンティファイ・メッセージ**を受け取ったときに**セーブされているポインタをリストア**する必要があります。

#### (3) ディスコネクト (04h)

このメッセージはターゲットがディスコネクトすることをイニシエータに知らせるためのメッセージです。出力後、ターゲットは**次のリセレクトまでバスをフリー**にします。

#### (4) イニシエータ・ディテクト・エラー (05h)

ターゲットからのデータ転送中にパリティ・エラーが発生したが、イニシエータがターゲットに**再データ転送のリトライ可能であることを知らせる**ためのメッセージです。

#### (5) アボート (06h)

イニシエータからターゲットに対して実行中の**動作の中止**を要求するメッセージです。ターゲットはアボート・メッセージを受け取ると実行中の動作を中断しバス・フリー・フェーズにします。

#### (6) メッセージ・リジェクト (07h)

イニシエータおよびターゲットから出力されるメッセージで、**直前に受け取ったメッセージが不適当であるか実行不可能である**ことを示します。

#### (7) ノー・オペレーション (08h)

ターゲットからのメッセージ・アウトの要求に対してイニシエータが**意味をもつメッセージをもたない場合にイニシエータが送出する**メッセージです。

#### (8) メッセージ・パリティ・エラー (09h)

イニシエータがターゲットに対して受理したメッセージにパリティ・エラーがあったことを知らせるためのメッセージで、**ターゲットはメッセージを再送**します。

#### (9) バス・デバイス・リセット (0Ch)

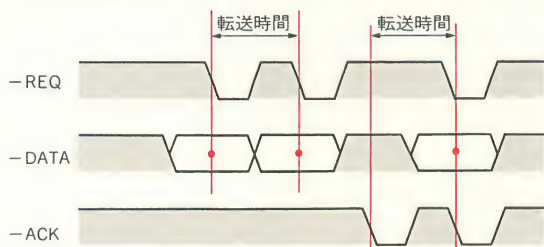
イニシエータがターゲットに対して実行中の命令を中断して**ターゲット・システムをリセット**し、バス・フリー・フェーズに戻ることを要求するメッセージです。



〈図 32〉 同期データ転送リクエスト

ビット バイト	7	6	5	4	3	2	1	0
0	拡張メッセージ (01H)							
1	拡張メッセージ長 (03H)							
2	同期データ転送リクエスト・コード (01H)							
3	転送時間 ( $m \times 4\text{ns}$ ) $m$							
4	REQ/ACK オフセット $X$							

〈図 33〉 同期データ転送の詳細(ターゲット→イニシエータ)



内部状態が初期化される点がアボート・メッセージと異なります。

#### (10) アイデンティファイ (80H, COH)

イニシエータおよびターゲットから出力されるメッセージで、**接続を確立**するためのメッセージです。

ビット 7 はアイデンティファイ・メッセージであることを示し、ビット 6 はディスクネクト/リセクション機能の有無を示すビットです。機能がある場合には“1”，ない場合には“0”となります。

ビット 5 からビット 3 は常に“0”です。

ビット 2 からビット 0 はターゲットの論理ユニット番号(LUN)です。

#### (11) 拡張メッセージ

拡張メッセージとしていくつかのメッセージが規定されていますが、ディスクで一般的に用いられる**同期データ転送リクエスト**を図 32 に示します。

同期データ転送機能をもつ SCSI 装置はイニシエータとターゲットの間で同期データ転送リクエストのメッセージを交換することで、転送時間と REQ/ACK オフセットを設定します。

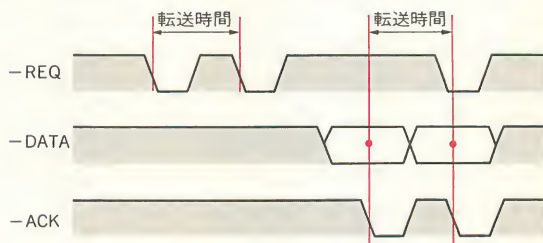
連続した -REQ もしくは -ACK の立ち下りの最小間隔を転送時間といいます。**ACK 信号**をターゲットが受信する前に送出することができる **REQ 信号**の最大数を REQ/ACK オフセットといいます。非同期モードとは REQ/ACK オフセットが 0 の場合です。オフセットが FFH の場合にはオフセット無制限を示します。

同期データ転送機能をもつイニシエータは最初にターゲットをセレクトしたときに -ATN を表明し、イニシエータが可能な REQ/ACK オフセットと最小転送時間を示す同期データ転送リクエストを送ります。

〈表 5〉 同期データ転送リクエスト(ターゲット応答)

ターゲット応答	結果
要求以下の REQ/ACK オフセット 要求時間以上の最小転送時間	ターゲット値と同じ REQ/ACK オフセット ターゲット値と同じ最小転送時間
REQ/ACK オフセットが 0 に等しい	非同期転送
MESSAGE REJECT	非同期転送

〈図 34〉 同期データ転送の詳細(イニシエータ→ターゲット)



これに対してターゲットは表 5 のいずれかの方法で応答し、結果として表に示したモードで転送することになります。

同期データ転送の詳細(オフセット 2 の場合)についてターゲットからイニシエータへの転送を図 33 に、イニシエータからターゲットへの転送を図 34 に示します。また転送時間についてもこれらの図中に示します。

図 33 においてターゲットは -REQ パルスに同期してデータを 2 個まで先に送出し、データを受け取ったイニシエータはターゲットに -ACK パルスを送出します。ターゲットは -ACK パルスを確認したとき、引き続きデータを送出しますが、-ACK パルスより先に送出できるデータの数は 2 個までです。

図 34 においてターゲットは -REQ パルスを 2 個まで先に送出し、データ転送を求めます。イニシエータはこれに対して -ACK パルスに同期してターゲットにデータを送ります。ターゲットは -ACK パルスを確認したとき -REQ パルスを必要な数だけ送出しますが、-REQ パルス数は -ACK パルス数と同じか多くても 2 個までです。

電源投入後、バス・デバイス・リセット・メッセージののち、あるいはハード・リセット条件の後では非同期モードです。同期データ転送リクエスト・メッセージ交換はイニシエータとターゲット両方においてセレクション・フェーズに続いて一回だけでなければなりません。

## SCSI バス信号

図 35 に SCSI バスのシングル・エンド型のピン・コネクションを示します。

基本的には SASI と同様ですが以下の各点で SASI

〈図 35〉 SCSI バス信号

1	GND	2	-DB <sub>0</sub>
3	GND	4	-DB <sub>1</sub>
5	GND	6	-DB <sub>2</sub>
7	GND	8	-DB <sub>3</sub>
9	GND	10	-DB <sub>4</sub>
11	GND	12	-DB <sub>5</sub>
13	GND	14	-DB <sub>6</sub>
15	GND	16	-DB <sub>7</sub>
17	GND	18	-DB <sub>P</sub>
19	GND	20	GND
21	GND	22	GND
23	GND	24	GND
25	N.C.	26	TERMPWR
27	GND	28	GND
29	GND	30	GND
31	GND	32	-ATN
33	GND	34	GND
35	GND	36	-BSY
37	GND	38	-ACK
39	GND	40	-RST
41	GND	42	-MSG
43	GND	44	-SEL
45	GND	46	-C/D
47	GND	48	-REQ
49	GND	50	-I/O

と異なります。

(1) -BSY(ビジー)

リセクション・フェーズにおいてイニシエータもドライブするようになりました。

またアービトレーション・フェーズにおける使用方法が追加されました。

(2) -SEL(セレクト)

リセクション・フェーズにおいてターゲットもドライブするようになりました。

また、アービトレーション・フェーズにおける使用方法が追加されました。

(3) -ATN(アテンション)

ターゲットはこのラインがアクティブ("L")であることを検出するとメッセージ・アウト・フェーズに入り、イニシエータからのメッセージを受け付けます。アテンションが("H")になったあとのハンドシェイクの終了からターゲットはそのメッセージ・データにしたがってつぎの動作を行います。

(4) TERMPWR(タームパワー)

終端抵抗電源を SCSI デバイスの外部に取り付ける場合、もしくは終端抵抗電源を実装した SCSI デバイスの電源が切断されることがあるシステムではこのラインに **SCSI デバイスから電源を供給する必要があります**。また電源供給にともないダイオードなどを使って逆流防止を施す必要があります、システムの電源容量が

〈図 37〉 SCSI コマンド一覧

コード	コマンド名	コード	コマンド名
00H	テスト・ユニット・レディ	(1BH)	(スタート/ストップ・ユニット)
01H	リゼロ・ユニット	25H	リード・キャパシティ
03H	リクエスト・センス	28H	リード・エクステンデッド
04H	フォーマット・ユニット	2AH	ライト・エクステンデッド
(07H)	(リアサイン・ブロック)	2BH	シーク・エクステンデッド
08H	リード	2EH	ライト&ベリファイ
0AH	ライト	2FH	ベリファイ
0BH	シーク	(37H)	(リード・ディフェクト・データ)
12H	インクワイアリー	C2H	セット・ディスク・パラメータ
15H	モード・セレクト	EOH	チェック・バッファ RAM
16H	リサーブ・ユニット	E1H	ライト・ループ・バック
17H	リリース・ユニット	E2H	リード・ループ・バック
1AH	モード・センス	E6H	リクエスト・エラー・ログ

〈図 36〉 ターミネータ・パワー供給回路



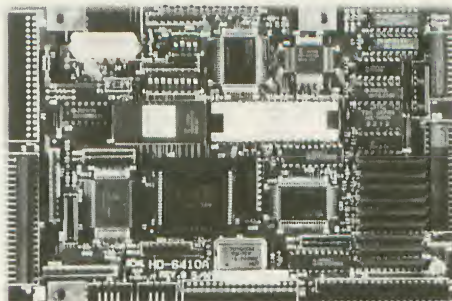
が大きく過電流がこのダイオードに流れる恐れのある場合などにはヒューズを取り付ける必要もあります(図 36)。

SCSI バスにはシングル・エンド型の他にディファレンシャル型があります。詳しくはコラムを参照してください。

SCSI も 1986 年に規格化されてから 5 年が経ち、その間のパーソナル・コンピュータや EWS に代表されるマイクロコンピュータの能力向上はかなりのものです。そこでデータ速度を向上させた **SCSI2** が SCSI の上位互換として規格化されました。これも基本的には SCSI と同じですので部分的には順次使用されていくと思われます(7 章参照)。

## SCSI コマンドの詳細

図 37 に示した各 SCSI コマンドの詳細を示します。パラメータなど機種依存性のある項目に対しては特に指定しない限り ICM 製 ESDI ディスク・コントローラ HD-6410A(写真 1)を例にとります。



〈写真 1〉 HD-6410A



〈図 38〉 テスト・ユニット・レディ

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 40〉 リクエスト・センス

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	アロケーション・レンジス							
5	0	0	0	0	0	0	0	0

(1) テスト・ユニット・レディー・コマンド (00H)

ターゲットが**レディ状態であるかどうかを調べるための命令**です(図 38)。この命令に対してターゲットはリード/ライト動作を含むすべての命令の実行が可能となきにはステータス・コードが正常終了(00H)。

〈図 39〉 リゼロ・ユニット

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 41〉 非拡張センス・データ・フォーマット

ビット バイト	7	6	5	4	3	2	1	0
0	AdrVal	センス・コード						
1	LUN			論理ブロック・アドレス (MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス (LSB)							

またそのほかのときにはチェック・コンディション(02H)となります。

(2) リゼロ・ユニット・コマンド (01H)

この命令を受けたターゲットは**装置内部のヘッド位置決め系の再初期化**を行い、**シリンドラ 0**へシークします(図 39)。

(3) リクエスト・センス・コマンド (03H)

## SCSI バスの電氣的条件

SCSI バスには 2 台から 8 台までの SCSI デバイスがディジィ・チェーンで接続され、両端にターミネータが接続されます。伝送系の電氣的条件としては本文中の説明で用いた**最大ケーブル長 6m**のシングル・エンド型のほかに**最大ケーブル長 25m**のディファレンシャル型があります。論理的信号のレベルではこれら 2 種類とも共通なのですが、**電氣的条件がまったく異なるので相互接続は不可能**です。

現在シングル・エンド型はドライバ、レシーバ内蔵型の SCSI コントローラを使用することが多く、外付けのドライバ、レシーバが必要なディファレンシャル型は汎用大型機用などきわめて限られたところでしか使用されていません。

### ● シングル・エンド型

すべての信号は負論理("L" でアクティブ)です。ドライバとしては 7438、レシーバとしては 74LS14 が電氣的条件を満たしています。

#### ▶ 出力特性

$V_{ol}$  ("L" レベル出力) = 0.0~0.4V  
 $(I_{ol}=4.8\text{mA}$  のとき  $V_{ol}=0.5\text{Vmax}$ )  
 $V_{oh}$  ("H" レベル出力) = 2.5~5.25V

#### ▶ 入力特性

$V_{il}$  ("L" レベル入力) = 0.0~0.8V  
 $(V_{il}=0.4\text{V}$  のとき  $I_{il}=0.4\text{mAmax}$ )  
 $V_{ih}$  ("H" レベル入力) = 2.0~5.25V  
 入力ヒステリシス = 0.2Vmin

### ● ディファレンシャル型

+信号と-信号の 2 本からなり、+信号の電位が-信号の電位より高いときアクティブ、-信号の電位が+信号の電位より高いときインアクティブです。マルチポイント伝送用である EIA 規格の RS485 に適合するドライバ、レシーバを用います。

#### ▶ 出力特性

$I_{ol}$  ("L" レベル出力) = 55mA のとき  
 $V_{ol}=2.0\text{Vmax}$   
 $I_{oh}$  ("H" レベル出力) = 55mA のとき  
 $V_{oh}=3.0\text{Vmax}$   
 $V_{od}$  (差動電圧) = 1.0Vmin

#### ▶ 入力特性

$I_t = \pm 2.0\text{mAmax}$   
 入力ヒステリシス = 35mVmin  
 コモン・モード電圧範囲 = -7V~+12V

ターゲットがチェック・コンディション・ステータスを送出した際にイニシエータが詳細情報を読み取るための命令です(図 40)。

HD-6410A では、図 42 の 22(16H)バイトの拡張センス・データをサポートします。機種によりこれ以上の長さのセンス・データをサポートするものもあります。それらのデータは機種固有のデータであり、一般的には 22 バイト目以降の追加部分を使用しないほうがいいでしょう。

図 40 のバイト 4 のアロケーション・レングスはイニシエータが必要とするセンス・データのバイト数を指定します。HD-6410A では 00H から 16H までが有効で、00H の場合は 4 バイトの非拡張センス・データ(SASI コマンドと互換を取るため同じフォーマット)を送出します(図 41)。非拡張センス・データについては SASI コマンドの詳細の項を参考にしてください。16H 以上の場合には 22 バイトとします(図 42)。

非拡張センス・データ(SASI)は論理ブロック・アドレスが 21 ビットなのに対し、拡張センス・データ(SCSI)は論理ブロック・アドレスを 32 ビットもっています。

そこで後述する拡張リード・コマンドのように 32 ビット・アドレスのコマンドを使用する場合には拡張センス・データを用いることになります。

センス・データのバイト 0 のビット 7 の **AdrVal** ビットは非拡張センス・データのバイト 1 からバイト 3

〈図 42〉 拡張センス・データ・フォーマット

ビット バイト	7	6	5	4	3	2	1	0
0	AdrVal	1	1	1	0	0	0	0
1	LUN			0	0	0	0	0
2	0	0	0	0	センス・キー			
3	論理ブロック・アドレス(MSB)							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス							
6	論理ブロック・アドレス(LSB)							
7	アディショナル・センス・レングス							
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12	センス・コード							
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0

まで、もしくは拡張センス・コードのバイト 3 からバイト 6 までの論理ブロック・アドレスが意味をもつ値であるかを示すビットで、AdrVal=“1”のときに論理ブロック・アドレスが意味をもつことを示します。AdrVal=“0”のときは論理ブロック・アドレスが意味をもたないことを示し、論理ブロック・アドレスのすべてのビットは 0 です。

バイト 2 のビット 3, 2, 1, 0 は異常の概要を示すセンス・キーで、表 6 に示す内容です。

バイト 12 は異常の詳細を示すセンス・コードで表 7 に示します。センス・コードは非拡張センス・データの場合のバイト 0 のビット 6 からビット 0 と同じコードです。

チェック・コンディション状態でない場合にリクエスト・センス・コマンドを受け取った場合はセンス・キー=0H、センス・コード=00H のノー・センスを送出します。

バイト 7 のアディショナル・センス・レングスはバイト 8 以降のセンス・データのバイト数を示します。

#### (4) フォーマット・ユニット・コマンド(O4H)

磁気ディスク媒体のフォーマットを要求する命令で

〈表 6〉 センス・キー

センス・キー	内 容
0H	No Sense : 装置に異常がなかったことを示す
1H	Recovered Error : 装置自体のリトライ機能によりコマンドが正常に終了したことを示す
2H	Not Ready : 装置自体に異常が発生し、オペレータ介入が必要であることを示す
3H	Medium Error : リード/ライト動作時において媒体に起因するエラーが発生し、リトライでエラーが回復できなかったことを示す
4H	Hardware Error : 装置のハードウェアに障害が発生し、コマンドの実行が正常終了しなかったことを示す
5H	Illegal Request : イニシエータからのコマンド・データに異常があり、コマンドの実行ができなかったことを示す
6H	Unit Attention : 装置がリセットされたことを示す
7H	Write Protected : ライト・プロテクト状態にあるときにライト動作を含むコマンドを受け取ったことを示す
BH	Aborted Command : ターゲットがイニシエータからのコマンドをアボートしたことを示す

〈図 43〉 フォーマット・ユニット・コマンド

バイト \ ビット	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0
1	LUN			データ CmpLst	ディフェクト・リスト・フォーマット			
2	0	0	0	0	0	0	0	0
3	インターリーブ(MSB)							
4	インターリーブ(LSB)							
5	0	0	0	0	0	0	0	0



す。HD-6410A はこの命令が送られても無視します  
が、パラメータ・リストは読み取ります。

図 43 のバイト 3 およびバイト 4 のインターリーブ  
はフォーマット時のインターリーブ数を示します。

バイト 1 のビット 4 からビット 0 でフォーマット命  
令実行時に使用する媒体欠陥を表すディフェクト・リ

ストの形式を設定します。リストの種類としては以下  
に示す 4 種のディフェクト・リストが使用されます。

① P-List(プライマリ・ディフェクト・リスト)

装置出荷前に工場内における媒体欠陥検査により検  
出された媒体欠陥リストで装置内の規定領域に書き込  
まれています。

〈表 7〉 センス・コード

センス・コード	センス・キー	ディスクリプション	センス・コード	センス・キー	ディスクリプション
00H	0H	No Sense : 装置に異常がなくコマンドが正常終了した	21H	5H	Illegal Block Address : ターゲットの持つブロック数より、大きな値の ブロックが指定された
01H	4H	No Index/Sector Signal : リード/ライト動作に必要なインデックス/セク タ信号を検出できなかった	24H	5H	Illegal Field in CDB : コマンド・ディスクリプタ・ブロックの 2 バイト 以降に不正なバイトがあった
02H	4H	No Seek Complete : シーク動作が正常に終了しなかった	25H	5H	Invalid LUN : 論理ユニットの指定が不適当である
03H	4H	Write Fault : ライト動作実行時、ライト系回路に異常があり、 ライト動作が正常終了しなかった	26H	5H	Invalid Field in Parameter List : コマンドに付随するパラメータに不正な内容が あった
04H	2H	Not Ready : 装置が起動動作中、もしくは装置に異常が発生 した	27H	4H	Write Protected : ライト・プロテクトに設定されている場合にラ イト動作を含むコマンドを受け取った
06H	4H	No Track 0 Found : リゼロ動作が正常に終了しなかった	29H	6H	Power on, Reset, Bus Device Changed : 装置が初期状態にある (SASI との関係で HD- 6410A では発生しない)
09H	4H	Track Following Error : ライト中に規定値以上のオフ・トラックが発生 した	2AH	6H	Mode Select Parameter Changed : 他のイニシエータにより、モード・セレクト・パ ラメータが変更された
10H	3H	ID CRC Error : 目的のブロックの ID 部の読み出し時に CRC エラーが発生し、回復できなかった	31H	3H	Medium Format Corrupted : 媒体欠陥によりフォーマットが正常終了しなかつ た
10H	4H	ID Read Error : 装置のリード回路に障害が発生し、ID 部が読 めない	32H	3H	No Defect Spare Location Available : 装置内のスペア・ロケーションがなくなり該当 ブロックの使用が不可能である
11H	3H	Unrecovered Read Error : リード時データ部にエラーが発生し、リトライ による回復ができなかった	40H	4H	RAM Error : 診断時、データ・バッファ RAM に異常があった
12H	3H	No ID Address Mark Found : ID 部のアドレス・マークが検出できず、ID 部 が検出できなかった	41H	4H	Data Path Diagnostic Failure : 診断時、ECC 回路に異常があった
13H	3H	No Data Address Mark Found : データ部のアドレス・マークが検出できず、デ ータ部の読み出しができなかった	42H	4H	Power On Diagnostic Failure : 電源投入時の自己診断で装置に異常があった。 他のセンス・コードで表示される異常について はそのコードが優先する
14H	3H	No Record Found : 目的のブロックを検出できなかった	43H	4H	Message Reject Error : イニシエータからメッセージ・リジェクトのメ ッセージを受け取った
15H	4H	Seek Error : シーク動作の異常が発生した	45H	4H	Select/Reselect Failure : リセレクトの不成功
17H	1H	Recovered Read Data with Retries : リード・リトライ動作により、データ部が読めた	47H	4H	SCSI Interface Parity Error : イニシエータからの情報にパリティ・エラーが あった
18H	1H	Recovered Read Data with ECC : ECC 機能によりエラー回復した	48H	BH	Initiator Detected Error : イニシエータから Initiator Detected Error Message を受け取った
19H	3H	Defect List Error : 欠陥リストの読み出し時エラーが発生し、正常 動作ができない	49H	BH	Inappropriate/Illegal Message : メッセージにおいて、装置が受け付けられない メッセージを受け取った
1AH	5H	Parameter Over Run : パラメータ転送時オーバ・ランが発生した			
20H	5H	Invalid Command Code : コマンド・ディスクリプタ・ブロックのコマン ド・コードが正常でない、もしくはターゲット がサポートしていないコマンドを受け取った			

〈図 44〉 ブロック・フォーマット  
(ヘッダおよび1 ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクト論理ブロック・アドレス(MSB)							
1	ディフェクト論理ブロック・アドレス							
2	ディフェクト論理ブロック・アドレス							
3	ディフェクト論理ブロック・アドレス(LSB)							

## ② C-List(ターゲット・サーティフィケーション・リスト)

フォーマット時に装置が検出した媒体欠陥リストです。

## ③ D-List(データ・ディフェクト・リスト)

フォーマット・コマンドのデータ・アウト・フェーズ時にイニシエータより送出される媒体欠陥リストです。

## ④ G-List(グロウン・ディフェクト・リスト)

P-List にない後発媒体欠陥リストです。

図 43 のバイト 1 のビット 4 からビット 0 と各ディフェクト・リストとの関係を以下に説明します。

ビット 4(データ)はフォーマット時にイニシエータが D-List をもつか否かを示すビットで“1”のときデータ・アウト・フェーズでディフェクト・データの受理を行います。ビット 3(Cmp Lst)は上記 D-List が装置のすべての媒体欠陥情報を含むか否かを示すビットで，“1”のときすべての媒体欠陥情報を含むことを示し，“0”のときは追加媒体欠陥情報であることを示します。

バイト 1 のビット 4, 3 の組み合わせにより下記の 3 種の方法でフォーマットします。

ビット 4, 3

0 0 : P-List と G-List によりフォーマット

1 0 : P-List, G-List および D-List によりフォーマット

1 1 : P-List と D-List によりフォーマット

D-List の形式にはブロック・フォーマット(論理ブロック・アドレスで欠陥位置を示すフォーマット)、バイト・フロム・インデックス・フォーマット(インデックス信号から欠陥までのバイト数で欠陥位置を示すフォーマット)、物理的セクタ・フォーマット(シリンダ、ヘッドとセクタ番号で欠陥位置を示すフォーマット)の 3 種類があります。図 43 のバイト 1 のビット 2, 1, 0 と各形式の対応を下記に示します。

ビット 2, 1, 0

0 × × : ブロック・フォーマット

〈図 45〉 バイト・フロム・インデックス・フォーマット  
(ヘッダおよび1 ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクトのシリンダ・ナンバ(MSB)							
1	ディフェクトのシリンダ・ナンバ							
2	ディフェクトのシリンダ・ナンバ(LSB)							
3	ディフェクトのヘッド・ナンバ							
4	インデックスからのディフェクト・バイト(MSB)							
5	インデックスからのディフェクト・バイト							
6	インデックスからのディフェクト・バイト							
7	インデックスからのディフェクト・バイト(LSB)							

(注) ディフェクト・ディスクリプタ(s)のバイト 4, 5, 6, 7 に示すインデックスからのディフェクト・バイトが FFFFFFFFH のときは該当シリンダ・ヘッドのすべてのブロックをリアサインすることを要求している

〈図 46〉 物理的セクタ・フォーマット  
(ヘッダおよび1 ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス(MSB)							
3	ディフェクト・リスト・レングス(LSB)							
0	ディフェクトのシリンダ・ナンバ(MSB)							
1	ディフェクトのシリンダ・ナンバ							
2	ディフェクトのシリンダ・ナンバ(LSB)							
3	ディフェクトのヘッド・ナンバ							
4	ディフェクト・セクタ・ナンバ(MSB)							
5	ディフェクト・セクタ・ナンバ							
6	ディフェクト・セクタ・ナンバ							
7	ディフェクト・セクタ・ナンバ(LSB)							

(注) ディフェクト・セクタ・ナンバが FFFFFFFFH の場合は該当シリンダ・ヘッドのすべてのブロックをリアサインすることを要求している

1 0 0 : バイト・フロム・インデックス・フォーマット

1 0 1 : 物理的セクタ・フォーマット

図 44, 図 45, 図 46 に各 D-List のフォーマットを示します。

D-List の先頭 4 バイトはディフェクト・リスト・ヘッダです。バイト 2, 3 にディフェクト・ディスクリプタのバイト数が示されます。ディフェクト・リスト・ヘッダ以降に媒体欠陥位置を示すディフェクト・ディスクリプタが 1 欠陥当たり 4 バイトもしくは 8 バイトで示されます。

## (5) リアサイン・ブロック・コマンド(O7H)

イニシエータがターゲットに対して指定のブロックのリアサイン(異なるロケーションへの再配置)を要求



〈図 47〉 リアサイン・ブロック・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 49〉 リアサイン・ブロック、ディフェクト・リスト  
(ヘッダおよび 1 ディフェクト分)

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	ディフェクト・リスト・レングス (MSB)							
3	ディフェクト・リスト・レングス (LSB)							
0	ディフェクト論理ブロック・アドレス (MSB)							
1	ディフェクト論理ブロック・アドレス							
2	ディフェクト論理ブロック・アドレス							
3	ディフェクト論理ブロック・アドレス (LSB)							

〈図 51〉 ライト・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0
1	LUN			論理ブロック・アドレス (MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス (LSB)							
4	転送長							
5	<i>r</i>	0	0	0	0	0	0	0

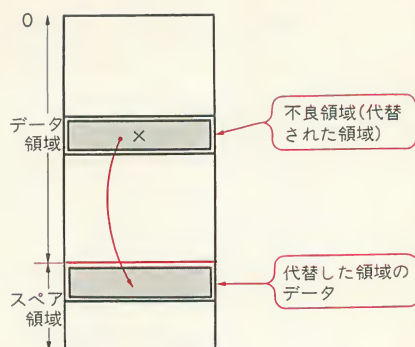
する命令です。ターゲットはあらかじめ確保されているスベア領域に指定のブロックを再配置します。ここでいうスベア領域とはデータ領域に不良領域がある場合に入れ換えてしまうことで正常な領域のように見せかけるための記憶領域です。この領域は図 48 に示すように一般にディスク・メディア上の内周側に割り当てられています。

本命令の実行後、指定のブロックを含むトラックをスベア・トラックに再配置します。ターゲットにディフェクト・スベアがなくリアサインできない場合にはノー・ディフェクト・スベア・ロケーション・アペイラブルのセンス・データをセットし、チェック・コンディション・ステータスを送出します。

図 49 にリアサイン・ブロック・コマンドでイニシエータから送出されるデータ形式を示します。データ形式はフォーマット・ユニット・コマンドのブロック・フォーマットと同一形式になります。

先頭 4 バイトはディフェクト・リストのバイト数を示すディフェクト・リスト・ヘッダで、バイト 2, 3 にディフェクト・ディスクリプタのバイト数を指定しま

〈図 48〉  
スベア領域



〈図 50〉 リード・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0
1	LUN			論理ブロック・アドレス (MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス (LSB)							
4	転送長							
5	<i>r</i>	<i>c</i>	0	0	0	0	0	0

〈図 52〉 シーク・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1
1	LUN			論理ブロック・アドレス (MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス (LSB)							
4	0	0	0	0	0	0	0	0
5	<i>r</i>	0	0	0	0	0	0	0

す。第 5 バイト以降はディフェクトの位置を示すディフェクト・ディスクリプタで、4 バイトで 1 個のディフェクトの論理ブロック・アドレスを示します。

#### (6) リード・コマンド (08H)

イニシエータがターゲットに対して磁気ディスク媒体に記録されているデータの転送を要求する命令です。

コマンド・ディスクリプタ・ブロックの論理ブロック・アドレスはデータ転送を要求する先頭のブロックを示し、転送長は転送を要求するブロック数を示します(転送長=00H は 256 個のブロックの転送要求を示す)。

なお図 50 中の *r* はリトライ禁止、*c* は ECC コレクション禁止のビットです。これらは SASI でよく使用されていたモード・ビットで、SCSI 規定ではベンダ・ユニーク・ビットとされています。HD-6410A では SASI との互換性のためこれらのビットをサポートしています。

#### (7) ライト・コマンド (0AH)

イニシエータがターゲットに対して磁気ディスク媒体へのデータの書き込みを要求する命令です。コマン

〈図 53〉 インクワイアリ・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0

ド・ディスクリプタ・ブロックの詳細はリード・コマンドと同じです。

#### (8) シーク・コマンド (0BH)

イニシエータがターゲットに対してコマンド・ディスクリプタ・ブロックの論理ブロック・アドレスで指定されるブロックの存在するシリンダにシークすることを要求する命令です (図 52)。

#### (9) インクワイアリ・コマンド (12H)

イニシエータがターゲットの属性を知るための命令です (図 53)。この命令に対しターゲットはインクワイアリ・データとして図 54 に示す 36 (24H) バイトを標準的に出力します。

図 53 のバイト 4 のアロケーション・レングスとしては 00H から 24H までの値が有効で、00H の場合にはインクワイアリ・データは転送されません。25H 以上の値が指定された場合には 36 バイトで転送を打ち切ります。

図 54 に示したインクワイアリ・データのバイト 0 はデバイス・タイプを示します。磁気ディスクは SCSI 規定のダイレクト・アクセス・デバイスに該当し、その場合の値は 0 です。バイト 2, 3 はターゲットが ANSI と CCS を満足していることを示します。バイト 4 のアディショナル・レングスはバイト 5 以降のバイト数を示し、インクワイアリ・データが 36 バイトのときには 31 (1FH) となります。バイト 8 からバイト 15 にはベンダ・アイデンティフィケーション “ICM”, バイト 16 からバイト 22 にはプロダクト・アイデンティフィケーション “HD-6410 A” (機種により固有の名前) を ASCII コードで示します。バイト 32 からバイト 35 はリビジョン・レベルを示す領域で、ターゲットのファームウェア・リビジョンを ASCII コードで示します。

図 54 のバイト 32 からバイト 35 はファームウェア・リビジョン 1.10 の場合の例を示します。

#### (10) モード・セレクト・コマンド (15H)

イニシエータがターゲットに対してモードの指定などを行うための命令です (図 55)。バイト 1 のビット 4 の PF (ページ・フォーマット) はモード・セレクト・ヘッダおよびブロック・ディスクリプタに続いてイニシエータから転送されるパラメータの形式を示すもの

〈図 54〉 インクワイアリ・データ

ビット バイト	7	6	5	4	3	2	1	0
0	ダイレクト・アクセス・デバイス・タイプ=00H							
1	0	0	0	0	0	0	0	0
2	ANSI スタンダード =01H							
3	CCS インプリメンテッド =01H							
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	“I” =49H							
9	“C” =43H							
10	“M” =4DH							
11	“ ” =20H							
12	“ ” =20H							
13	“ ” =20H							
14	“ ” =20H							
15	“ ” =20H							
16	“H” =48H							
17	“D” =44H							
18	“—” =2DH							
19	“6” =36H							
20	“4” =34H							
21	“1” =31H							
22	“0” =30H							
23	“ ” =20H							
24	“A” =41H							
25	“ ” =20H							
26	“ ” =20H							
27	“ ” =20H							
28	“ ” =20H							
29	“ ” =20H							
30	“ ” =20H							
31	“ ” =20H							
32	ファームウェア・リビジョン “0” =30H							
33	ファームウェア・リビジョン “1” =31H							
34	ファームウェア・リビジョン “1” =31H							
35	ファームウェア・リビジョン “0” =30H							

で、“1” の場合、CCS Rev. 4B にしたがつていることを示します。

HD-6410A の場合には “1”, “0” とともに以下に示すページ・フォーマットとして命令を実行します。

バイト 1 のビット 0 の SP (セーブ・パラメータ) はモード・セレクト・コマンドで転送されるパラメータの取り扱いを指定するビットです。バイト 4 のパラメータ・リスト・レングスはパラメータのバイト数を示します。

モード・セレクト・コマンドのパラメータにはモード・セレクト・ヘッダ、ブロック・ディスクリプタおよび数種のページ・パラメータがあります (図 56, 図 57, 図 58)。HD-6410A ではページ・パラメータとし



〈図 55〉 モード・セレクト・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1
1	LUN			PF	0	0	0	SP
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	パラメータ・リスト・レングス							
5	0	0	0	0	0	0	0	0

〈図 57〉 モード・セレクト・ブロック・ディスクリプタ

ビット バイト	7	6	5	4	3	2	1	0
0	デンシティ・コード=00H							
1	ブロック・ナンバ(MSB)=00H							
2	ブロック・ナンバ=00H							
3	ブロック・ナンバ(LSB)=00H							
4	0	0	0	0	0	0	0	0
5	ブロック・レングス(MSB)=00H							
6	ブロック・レングス=01H, 02H							
7	ブロック・レングス(LSB)=00H							

て、

1H：エラー・リカバリ・パラメータ

3H：ダイレクト・アクセス・デバイス・フォーマット・パラメータ

4H：リジッド・ディスク・ドライブ・パラメータの各パラメータをサポートします。詳細については後述します。

図 56 にモード・セレクト・ヘッダを示します。

バイト 3 の **ブロック・ディスクリプタ・レングス** は、モード・セレクト・ヘッダに引き続いて転送されるモード・セレクト・ブロック・ディスクリプタ(図 57)のバイト数を示すものです。ここが 8 の場合、次の 8 バイトがブロック・ディスクリプタです。0 の場合にはブロック・ディスクリプタはありません。

図 57 に示すブロック・ディスクリプタのバイト 0 はデンシティを規定するものですが、HD-6410A の

〈図 56〉 モード・セレクト・ヘッダ

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	ブロック・ディスクリプタ・レングス=00H または 08H							

〈図 58〉 エラー・リカバリ・パラメータ

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	ページ・コード=1H					
1	ページ・レングス=(06H)							
2	0	0	TB	RC	EEC	PER	DTE	DCR
3	リトライ・カウント							
4	コレクション・スパン							
5	ヘッド・オフセット・カウント							
6	データ・ストローブ・オフセット・カウント							
7	リカバリ・タイム・リミット							

場合は固定です。バイト 1 からバイト 3 のブロック・ナンバは**装置当たりのブロック数を規定する領域です。ブロック・レングス、セクタ/トラックの値によりターゲットの中で自動的に決定されるので、設定はできません。**バイト 5 からバイト 7 のブロック・レングスはブロック当たりのバイト数を指定する領域ですが HD-6410A の場合にはボード上のディップ・スイッチで決定されるため設定できません。

モード・セレクト・ヘッダ、ブロック・ディスクリプタに引き続いてページ・ディスクリプタが転送されます。

図 58 に**エラー・リカバリ・パラメータ**を示します。バイト 0 のページ・コードは本パラメータ群がエラー・リカバリ・パラメータであることを示します。バイト 1 のページ・レングスはバイト 2 以降の本パラメータ群のバイト数を示すものです。バイト 2 はリトライ方法の詳細を示すものです。

表 8 に EEC, PER, DTE, DCR ビットの組み合わせ

〈表 8〉 リトライ方法の詳細

EEC	PER	DTE	DCR	動作内容
0	0	0	0	リトライを行った後に ECC 修正を行い、修正後正常終了する。修正不可のときチェック・コンディション・ステータスで異常終了
0	0	0	1	ECC 修正を行わない。他の動作は上と同じ
0	1	0	0	リトライ後、ECC 修正し、修正可能な場合、すべてのデータの転送後、チェック・コンディション・ステータス、リカバード・センス・キーをセットして修正
0	1	0	1	ECC 修正を行わない。他は上と同じ動作
0	1	1	0	リトライを行った後、ECC 修正を実行し、チェック・コンディション・ステータス、リカバード・センス・キーをセットしデータ転送を中止
0	1	1	1	ECC 修正を行わない。他は上と同じ動作
1	0	0	0	ECC 修正を行い、修正不可能の場合リトライを実行。他は 0000 と同じ
1	1	0	0	ECC 修正を行い、修正不可能の場合リトライを実行。他は 0100 と同じ
1	1	1	0	ECC 修正を行い、修正不可能の場合リトライを実行。他は 0110 と同じ

〈図 59〉 リザーブ・ユニット・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0
1	LUN			3rdpty	サード・パーティ・デバイス ID			0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 61〉 モード・センス・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0
1	LUN			0	0	0	0	0
2	PCF		ページ・コード					
3	0	0	0	0	0	0	0	0
4	アロケーション・レングス							
5	0	0	0	0	0	0	0	0

わせによる動作内容を示します。

▶ DCR(ディスエイブル・コレクション)：エラー修正に ECC 機能を使用するか否かを指定するビットです。“1” のときは ECC 機能を使用することを禁止し，“0” のとき ECC 機能の使用を許可します。指定がない場合には ECC 機能を用いての修正を行います。

▶ DTE(ディスエイブル・トランスファ・イン・エラー)：エラー発生時点でのイニシエータへのデータ転送について指定するビットです。

▶ PER(ポスト・エラー)：エラー発生時のエラー発生状況の報告方法について規定するビットです。

▶ EEC(イネーブル・アーリ・コレクション)：リトライと ECC 機能の使用の順序を指定するビットです。

▶ RC(リード・コンティニュー)：“1” のとき、エラー発生時にリトライ/ECC 修正を行うことなくデータの転送を指示するビットです。指定がない場合にはリトライ/ECC 修正を行います。

▶ TB(トランスファ・ブロック)：“1” のとき、エラー発生に関係なくデータ転送を指示するもので、“0” のときエラー発生ブロックは転送されません。指定がない場合はデータを転送します。

バイト 3 のリトライ・カウントは ECC 機能による修正前に実行するリトライ数を指定するバイトです。ターゲットでは標準を 8 回とします。なお、F<sup>2</sup>F<sup>2</sup>H が指定された場合には無限回となります。

#### (11) リザーブ・ユニット・コマンド (16H)

複数のイニシエータがある場合にイニシエータがターゲットを占有するために用いる命令です(図 59)。この命令によって特定のイニシエータから占有されている場合、ほかのイニシエータからコマンドを受けてもリザーベーション・コンフリクト・ステータスにより命

〈図 60〉 リリース・ユニット・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1
1	LUN			3rdpty	サード・パーティ・デバイス ID			0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

〈図 62〉 エラー・リカバリ・パラメータの例  
(変更可能ビットのレポート)

ビット バイト	7	6	5	4	3	2	1	0
0	1	ページ・コード=1H						
1	ページ・レングス=06H							
2	0	0	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1

令の実行を拒否します。イニシエータ自体がリザーブするユニット・リザーベーションとほかの装置への占有を命令するサード・パーティ・リザーベーションがあります(マルチプロセスのイニシエータが同一のターゲットを複数のプロセスから同時にアクセスする可能性がある場合にプロセス間でターゲットをロックするために使用されるイクステンデッド・リザーベーションはサポートしない)。

図 59 のバイト 1 のビット 4 は本命令がサード・パーティ・リザーベーションであるか否かを示すビットで、“1” のときサード・パーティ・リザーベーションであることを示し、ビット 3 からビット 1 の 3 ビットで占有されるべきイニシエータの ID(000~111)を示します。

#### (12) リリース・ユニット・コマンド (17H)

この命令は上で説明したリザーブ・ユニット・コマンドによるターゲットの占有を解除するための命令で、リザーブを行ったイニシエータからのリリース・ユニット・コマンドのみ有効です(図 60)。コマンド・ディスクリプタ・ブロックのバイト 1 の意味はリザーブ・ユニット・コマンドと同じです。

#### (13) モード・センス・コマンド (1AH)

イニシエータがターゲットの各種パラメータを知るための命令です(図 61)。

バイト 2 のビット 5 からビット 0 のページ・コードは転送するセンス・データの内容を示します。バイト 4 のアロケーション・レングスはセンス・データを受け取るためにイニシエータが用意している領域のバイト数を示します。バイト 2 のビット 7, 6 の PCF(ページ・コントロール・フィールド)はターゲットから転送



〈図 63〉 モード・センス・ヘッダ

バイト	ビット	7	6	5	4	3	2	1	0
0	センス・データ・レングス								
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	ブロック・ディスクリプタ・レングス=08H								

〈図 64〉 モード・センス・ブロック・ディスクリプタ

バイト	ビット	7	6	5	4	3	2	1	0
0	デンシティ・コード=00H								
1	ブロック・ナンバ(MSB)								
2	ブロック・ナンバ								
3	ブロック・ナンバ(LSB)								
4	0	0	0	0	0	0	0	0	0
5	ブロック・レングス(MSB)								
6	ブロック・レングス								
7	ブロック・レングス(LSB)								

〈図 65〉 エラー・リカバリ・パラメータ

バイト \ ビット	7	6	5	4	3	2	1	0
0	1	ページ・コード=1H						
1	ページ・レングス=06H							
2	0	0	TB	RC	EEC	PER	DTE	DCR
3	リトライ・カウント=08H							
4	コレクション・スパン=08H							
5	ヘッド・オフセット・カウント=00H							
6	データ・ストローブ・オフセット・カウント=00H							
7	リカバリ・タイム・リミット=F'FH							

するパラメータのタイプを示すもので、以下の 4 種があります。

- ① ビット 7, 6=0, 0 の場合：現在値のレポート  
モード・セレクト・コマンドで指定された現在の値を転送します。
- ② ビット 7, 6=0, 1 の場合：変更可能ビットのレポート  
センス・データの変更可能なビットおよびフィールドを“1”とし、変更不可能なビットおよびフィールドを“0”としたデータを転送します。
- エラー・リカバリ・パラメータ・ページにおける変更可能ビットの例を図 62 に示します。図 58 に示した内容で変更可能な部分を 1 で示しています。
- ③ ビット 7, 6=1, 0 の場合：規定値のレポート  
センス・データのデフォルト値を転送します。
- ④ ビット 6, 7=1, 1 の場合：保存されている値のレポート  
センス・データのフォーマット・ユニット・コマンドが完了している値を転送します。

ページ・コードの値が 3FH の場合はページ番号順

〈図 66〉 ダイレクト・アクセス・デバイス・フォーマット・パラメータ

ビット バイト	7	6	5	4	3	2	1	0
0	1	ページ・コード=3H						
1	ページ・レングス=16H							
2	トラック/シリンダ(MSB)=00H							
3	トラック/シリンダ(LSB)=XXH							
4	代替セクタ/シリンダ(MSB)=00H							
5	代替セクタ/シリンダ(LSB)=XXH							
6	代替セクタ/シリンダ(MSB)=00H							
7	代替セクタ/シリンダ(LSB)=00H							
8	代替セクタ/ユニット(MSB)=00H							
9	代替セクタ/ユニット(LSB)=XXH							
10	セクタ/トラック(MSB)=00H							
11	セクタ/トラック(LSB)=XXH							
12	データ・バイト/物理セクタ(MSB)=00H							
13	データ・バイト/物理セクタ(LSB)=00H							
14	インターリーブ(MSB)=00H							
15	インターリーブ(LSB)=01H							
16	トラック・スキュー・ファクタ(MSB)=00H							
17	トラック・スキュー・ファクタ(LSB)=00H							
18	シリンダ・スキュー・ファクタ(MSB)=00H							
19	シリンダ・スキュー・ファクタ(LSB)=00H							
20	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0

にターゲットがサポートしているすべてのページのデータが転送されます。ページ・コードの値が 3FH 以外の場合は指定されたページのデータが転送されます。

次にターゲットがサポートするセンス・データのページについて説明します。ターゲットが磁気ディスク装置の場合、下記の 3 種類のページをサポートするのが一般的です。

1h：エラー・リカバリ・パラメータ

3h：ダイレクト・アクセス・デバイス・フォーマット・パラメータ

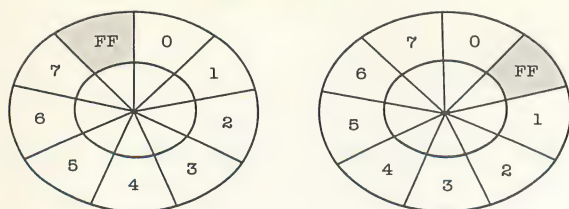
4h：リジッド・ディスク・ドライブ・ジオメトリ・パラメータ

ページ・コードが 3FH の場合には、モード・センス・ヘッダ、ブロック・ディスクリプタ、エラー・リカバリ・パラメータ、ダイレクト・アクセス・ドライブ・フォーマット・パラメータ、リジッド・ディスク・ドライブ・ジオメトリ・パラメータの順に転送します。

モード・センス・コマンドに対してターゲットはデータ・イン・フェーズで図 63 に示すモード・センス・ヘッダを転送し、つぎに図 64 に示すモード・センス・ブロック・ディスクリプタを転送します。

モード・センス・ヘッダのバイト 0 のセンス・データ・レングスは、モード・センス・コマンドによりデー

〈図 67〉 代替セクタ



(a) 不良スポットがない場合

(b) 不良スポットがある場合

〈図 68〉 リジッド・ディスク・ドライブ・ジオメトリ・パラメータ

ビット バイト	7	6	5	4	3	2	1	0
0	0	ページ・コード=4H						
1	ページ・レングス=12H							
2	シリンダ・ナンバ(MSB)=00H							
3	シリンダ・ナンバ =XXH							
4	シリンダ・ナンバ(LSB)=XXH							
5	ヘッド・ナンバ =XXH							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0

タ・イン・フェーズでイニシエータに転送されるセンス・データのバイト数を示し、(全バイト数-1)の値です。これはページ・コードの指定によって異なります。バイト 3 はブロック・ディスクリプタのバイト数 08H を示します。

モード・センス・ブロック・ディスクリプタは図 64 に示すような 8 バイトで構成されます。バイト 0 のデンシティ・コードは磁気ディスクではデフォルト値は 00H です。バイト 1 からバイト 3 のブロック・ナンバはターゲットで使用可能なブロック総数を示し、バイト 5, 6, 7 のブロック・レングスは 1 ブロック当たりのバイト数を示す領域です。

つぎにそれぞれのページについて説明します。

図 65 に示すエラー・リカバリ・パラメータはターゲットにおけるエラー・リトライの条件を示します。各バイト、ビットの意味はモード・セレクト・コマンドの項を参照してください。

図 66 にダイレクト・アクセス・デバイス・フォーマ

〈図 69〉 スタート/ストップ・ユニット・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1
1	LUN			0	0	0	0	IMMED
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	ST/SP
5	0	0	0	0	0	0	0	0

〈図 70〉 リムーブ可/不可コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0
1	LUN			0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	禁止
5	0	0	0	0	0	0	0	0

ット・パラメータを示します。バイト 2 からバイト 9 はハンドリング・オブ・ダイレクト・フィールドの領域です。HD-6410A では各トラックに 1 ないし 2 個の代替セクタを準備し、ディフェクトに対しては同一トラック内でのアロケーションを行います。

トラックをフォーマットしたときに図 67 (a) に示すように不良セクタがない場合にはいちばん最後のセクタを仮の不良セクタとして使用しません。不良セクタがある場合、不良セクタのところを FF として飛ばしてセクタを割り付け、有効セクタがすべて正常にアクセスできるようにします。代替セクタ数よりも不良セクタの数が多い場合には同一トラック内へのアロケーションができません。この場合には代替トラックへのアロケーションを行います。

バイト 10, 11 はトラック当たりのセクタ数を示す領域です。HD-6410A では代替セクタを含まない値を示します。機種(メーカ)によっては代替セクタを含む総セクタ数を示す場合があります。

バイト 12 からバイト 19 のセクタ・フォーマット・フィールドはフォーマット・ユニット時のパラメータを示し、1 セクタ当たりのバイト数により決まる値が転送されます。

図 68 のリジッド・ディスク・ドライブ・ジオメトリ・パラメータはターゲットのシリンダ数、ヘッド数を示します。ほかのバイトについては 00H を転送します。XXH はディスクによってパラメータが変化することを示します。

(14) スタート/ストップ・ユニット・コマンド(1BH)

イニシエータがターゲットに対して装置(スピンドル・モータ)の起動/停止を要求する命令です(図 69)。バイト 4 のビット 0 の ST/SP ビットが、



〈図71〉 リード・キャパシティ・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	1
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0

〈図72〉 リード・キャパシティ・データ

ビット バイト	7	6	5	4	3	2	1	0
0	論理ブロック・アドレス (MSB)							
1	論理ブロック・アドレス							
2	論理ブロック・アドレス							
3	論理ブロック・アドレス (LSB)							
4	ブロック・レングス (MSB)							
5	ブロック・レングス							
6	ブロック・レングス							
7	ブロック・レングス (LSB)							

ST/SP= “1”：スタート・ユニット

ST/SP= “0”：ストップ・ユニット

という意味をもちます。

バイト1のビット0はイミディエット (IMMED) ・ビットを示し、“1”のとき本命令の起動準備が完了した時点でステータスを送出しコマンドを終了します。“0”の場合は本命令の実行が完了するまで終了しません。

なおターゲットは電源 ON 時に本命令とは関係なくスピンドル・モータを起動します。

#### (15) リムーブ可/不可コマンド (1EH)

ターゲットに論理ユニットの媒体の取り外しを可能、不可能とすることを要求します (図70)。

バイト4のビット0が“1”のときは媒体の取り外しを禁止し、“0”のときは媒体の取り外しを許可します。MO (光磁気ディスク) の場合などに有効なコマンドです。

#### (16) リード・キャパシティ・コマンド (25H)

図72に示すリード・キャパシティ・データのバイト0からバイト3にターゲットの最大論理ブロック・アドレスを、バイト4からバイト7にそのブロック・レングスを出力します (図71)。

#### (17) 拡張リード・コマンド (28H)

イニシエータがターゲットに対して磁気ディスク媒体に記録されているデータの転送を要求する命令です (図73)。リード・コマンド (08H) とは論理ブロック・

〈図73〉 拡張リード・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	0	0	0	0	0	0	0	0
7	転送長 (MSB)							
8	転送長 (LSB)							
9	r	c	0	0	0	0	0	0

〈図74〉 拡張ライト・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	0
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	0	0	0	0	0	0	0	0
7	転送長 (MSB)							
8	転送長 (LSB)							
9	r	0	0	0	0	0	0	0

〈図75〉 拡張シーク・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	1
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0

アドレスと転送長の指定範囲が異なります。

ブート・ストラップ・ロードのようなプログラムでは SASI コマンドのみの HD でも使用できるように 08H のリード・コマンドを使用することがありますが、SCSI デバイスであることが確定している場合には通常 28H のリード・コマンドを使います。本命令の転送長が 0000H の場合はデータ転送の要求がないことを示します。

#### (18) 拡張ライト・コマンド (2AH)

イニシエータがターゲットに対して磁気ディスク媒体上へのデータ書き込みを要求する命令です (図74)。

〈図 76〉 ライト・アンド・ベリファイ・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	0
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	0	0	0	0	0	0	0	0
7	転送長 (MSB)							
8	転送長 (LSB)							
9	0	0	0	0	0	0	0	0

ライト・コマンド (OAH) とは論理ブロック・アドレスと転送長の指定範囲が異なります。転送長の指定が 0000H の場合は書き込みがないことを示します。

#### (19) 拡張シーク・コマンド (2BH)

イニシエータがターゲットに対してバイト 2 からバイト 5 で示される論理ブロック・アドレスで指定されるブロックの存在するシリングにシークすることを要求する命令です (図 75)。シーク・コマンド (OBH) とは論理ブロック・アドレスの指定範囲が異なるだけで、同じ動作をします。

#### (20) ライト・アンド・ベリファイ・コマンド (2EH)

イニシエータがターゲットに対して、磁気ディスク媒体上へのデータの書き込みと書き込みデータの確認を要求する命令です (図 76)。ターゲットでは ECC のチェック機能により書き込みデータの確認を行います。コマンド・ディスクリプタ・ブロックの論理ブロック・アドレスと転送長の意味は拡張ライト・コマンド (2AH) と同じです。

#### (21) ベリファイ・コマンド (2FH)

イニシエータがターゲットに対して磁気ディスク媒

〈図 77〉 ベリファイ・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1	1
1	LUN			0	0	0	0	0
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	0	0	0	0	0	0	0	0
7	ベリフィケーション・レングス (MSB)							
8	ベリフィケーション・レングス (LSB)							
9	0	0	0	0	0	0	0	0

体上に書き込まれているデータの確認を要求する命令です (図 77)。ターゲットは ECC のチェック機能によってデータ・ベリファイを実行します。

以上でハードディスクにおいて通常定義されているコマンドを説明してきました。しかし、これらのコマンドが SCSI ディスクにおいてすべて共通で使用できるというわけでもありません。

できるだけ不特定多数の SCSI ハード・ディスクを共通のソフトウェアで使用することを前提とするならばマッキントッシュの OS が行っているようにリード・キャパシティ (25H)、拡張リード (28H)、拡張ライト (2AH) とリクエスト・センス (03H) の 4 種類の SCSI コマンドのみを使用することを推奨します。

#### ●参考・引用文献●

- (1)\*Small Computer Interface(SCSI) ANSI X3.131-1986, ANSI, 1986.
- (2) 最新 SCSI マニュアル, 第 2 版, CQ 出版社, 1989.
- (3) 事例にみる SCSI インターフェース技術, トリケップス, 1989.



別冊 インターフェース

好評発売中!

# 最新SCSIマニュアル

標準入出力インターフェースの規格・使い方・設計ノウハウ

1987年7月号の本誌 SCSI 特集号に、その後の動向をふまえて、大幅な加筆と全面的な改訂をおこなった SCSI の決定版。SCSI-2 の最新動向も収録。

2色刷 B5版 192ページ  
定価1,650円(税込み)  
送料260円 **CQ出版社**



# 代表的な SCSI コントローラ

松村清明

## ● WD33C93/WD33C93A(ウェスタン・デジタル)

SCSI コマンドをフルサポートしたインテリジェント・チップとして最初に出荷されたものです。内部のマイクロプロセッサがシーケンスをコントロールするので、一連の動作をひとつのコマンドで実行できます(図B)。

### ▶特徴

- ・SCSI をフルサポート：アービトレーション(ARBITRATION), ディスコネクト(DISCONNECT), リコネクト(RECONNECT), パリティ(PARITY), 同期式のデータ転送で4Mバイト/sで転送可能。オフセットは1から5までプログラムで選択可能。
- ・対ホスト用のデータ・バスは8ビット長であり、アドレスとデータ・バスがマルチプレクサされたシステム, またはアドレスとデータ・バスがマルチプレクサされていないシステムのどちらにも対応できる。
- ・ホスト・アダプタまたはペリフェラル・アダプタのどちらにも使用できる。
- ・WD33C93 には48mAのドライバを含んでおり, SCSIバスに直結可能。
- ・プログラムド I/O, DMA 転送またはダイレクト・バッファ・アクセスのいずれかを選択可能。
- ・セクションおよびセクション用のプログラム・タイマ内蔵。
- ・“コンビネーション” コマンドはインタラプト・ハンドリングの負担を大いに軽減する。
- ・特別な“トランスレイト・アドレス” コマンドはロジカル・アドレスからフィジカル・アドレスへの変更を可能にする。

・24ビットの転送カウンタ内蔵。

・+5VDC 単一電源。

・CMOS

## ● NCR5380

マッキントッシュで使用されている SCSI チップとして有名。データの転送を DMA で行えるロジック以外はプログラム I/O でコントロールする必要があります。その分シンプルで, CPU が直接コントロールするならオーバ・ヘッドが少ないといえます。

### ▶特徴

SCSI インターフェース

- ・非同期 1.5MBPS
- ・イニシエータ, ターゲット・サポート
- ・パリティ発生/チェック(オプション)
- ・アービトレーション・サポート
- ・すべてのバス信号直接制御
- ・バス・ドライバの内蔵

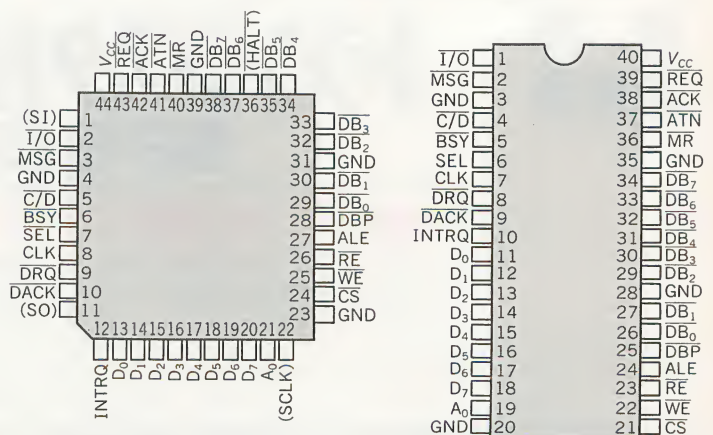
MPU インターフェース

- ・メモリ, I/O マップ・インターフェース
- ・DMA/プログラムド I/O
- ・ノーマル, ブロック・モード DMA
- ・MPU 割り込み(オプション)

## ● MB87034

MB87034 は, 富士通の一連の(同一コマンド体系の)SCSI コントローラの中の新しいチップです。ヒューレットパッカード社のワークステーションなどで使用されているインテリジェント・コントローラです。-REQ, -ACK 端子に3ステート・タイプのドライバ/レシーバを採用し, 伝送特性の改善をしてい

〈図B〉  
WD33C93



(a) 44ピン・フラット

(b) 40ピンDIP

ます。

#### ▶特徴

- ・ANSI XC3.131-1986 SCSI サポート
- ・48mA シングル・エンド型 ドライバ/レシーバ内蔵
- ・-REQ, -ACK 端子にトライステートのドライバ/レシーバ採用
- ・イニシエータ機能およびターゲット機能の両方をサポート
- ・同期モード転送による最高4M バイト/s までのデータ転送が可能
- ・同期モード転送速度は4段階プログラマブル
- ・同期モード転送オフセット値は1~8 バイト・プログラマブル
- ・8 バイトのデータ・バッファ・レジスタを内蔵
- ・28 ビット長の転送バイト・カウントにより、一度に256M バイトのデータ転送が可能
- ・独立したデータ転送用バス
- ・2 系統の割り込み信号
- ・+5V 単一電源、低消費電力動作
- ・入出力は TTL コンパチブル

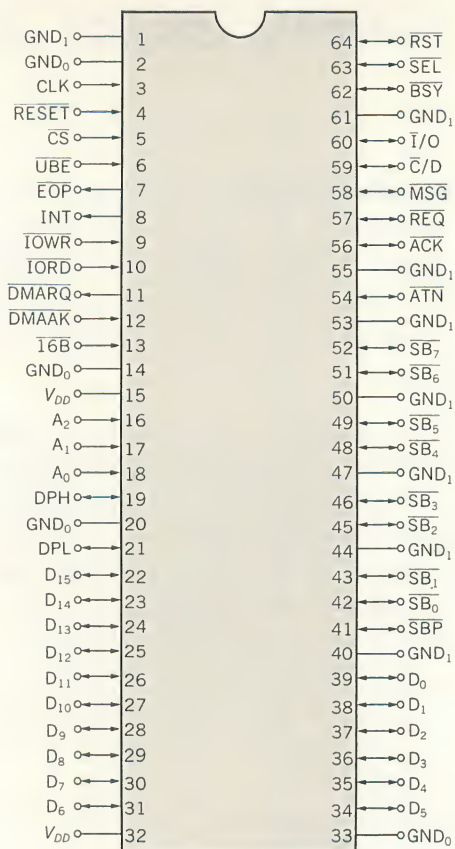
#### ●μPD72111

μPD72111 は今回実験に使用した SCSI コントローラです。16 ビット・バスで DMA 可能であり、SCSI バスのフルスピード連続転送を容易に実現できます(図 C)。

#### ▶特徴

- ・ANSI X3.131-1986 に準拠
- ・2 種類のデータ転送機能  
同期転送(最大 4.0M バイト/s: 1 から 8 までのオフセット値を指定可能)  
非同期転送(最大 1.5M バイト/s)
- ・イニシエータおよびターゲットとして動作可能
- ・シングル・エンド・タイプの SCSI バス駆動用ドライ

〈図 C〉 μPD72111



バおよびシュミット・タイプ・レシーバ内蔵 CPU 側のバス幅を選択可能(16 ビット/8 ビット)

- ・プログラム転送または DMA 転送を選択可能
- ・24 ビット転送カウンタ内蔵
- ・FIFO タイプのデータ・バッファを SCSI バス側および CPU 側の両方に内蔵

好評  
発売中

# トランジスタ技術 SPECIAL No.9

B5判 160頁  
定価1,540円  
(税込み)  
送料 260円

特集 パソコン周辺機器インターフェース詳解  
セントロニクス/RS-232C/GPIB/SCSIを理解するために

パソコン/マイコンが外界と通信するために必要な代表的四大インターフェースを詳しく解説します。まず最初に、インターフェースの規格について説明します。そして、そのインターフェースを実現するのに使われるLSI/ICの使い方を説明します。後半では、説明だけでは理解にくいタイミングなどを、実際の製作記事をとおして学んでいきます。

CQ出版社

〒170 東京都豊島区巣鴨1-14-2

☎03-5395-2121(出版部)  
03-5395-2141(営業部)

振替 東京0-10665



## 第5章

# SCSI を使ってハードディスクをつなぐ

ハードディスクを使うための作業はこれだけ

松村清明

それでは実際に SCSI コントローラを PC9801 に実装してハードディスクを接続します。次にこのディスクを使用するためのソフトウェアについて説明し、実際に MS-DOS のディスクとして使用する方法を述べます。

### 実験装置

#### ● SCSI コントローラ

SCSI コントローラ(イニシエータ)・ボードとして NEC 製の ET0018(写真 1)を使用しました。これは  $\mu$ PD72111(NEC 製 SCSI コントローラ)の評価用ボードです。

ET0018 のブロック図を図 1 に、回路図を図 2 に示します。

回路図にはバンク・メモリも記述されていますが、使用しないのでブロック図ではこれをはぶいて説明します。

$\mu$ PD72111 は SCSI バス・ドライバ内蔵の SCSI コントローラですから直接 SCSI バスに接続します。

拡張バス・インターフェースから見ると  $\mu$ PD72111 と I/O コントロール用のレジスタ(IC<sub>19</sub>および IC<sub>26</sub>)が I/O デバイスとして見えます。I/O アドレスはデコード回路(IC<sub>10</sub>および IC<sub>11</sub>)により決定されます。

その他、本回路では DMA として DRQ<sub>30</sub> と DACK<sub>30</sub> を、割り込みとして IR<sub>31</sub> を使用します。

#### ● 実験装置の接続

SCSI ハードディスクとして ICM 製 SRC-41(写真 2)を接続しました。PC9801 を含めた実験装置の接続を図 3 に示します。

ET0018 の設定を図 4 で説明します。

ジャンパ・スイッチ JP<sub>1</sub>, JP<sub>2</sub> はバンク・メモリの使用方法を設定しますが、今回は使用しません。

ジャンパ・スイッチ JP<sub>3</sub> は  $\mu$ PD72111 の INT 端子出力をハードウェア割り込みとして使用するか、ハードウェア割り込みを禁止するかを設定するものです。今回は INT 端子出力を割り込みとして使用します。具体的には 1-2 をショートします。

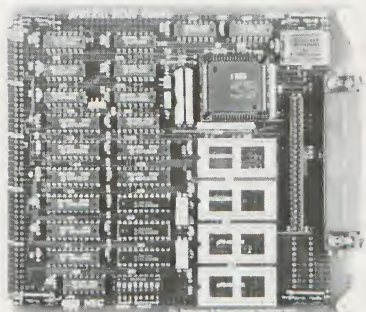
ジャンパ・スイッチ JP<sub>4</sub> は SCSI バス上の TER MPWR に電源供給するか否かを設定します。今回は電源供給をしません。オープンです。

ジャンパ・スイッチ JP<sub>5</sub>, JP<sub>6</sub> はバンク・メモリを使用するかどうかの設定です。今回は使用しないので JP<sub>5</sub>, JP<sub>6</sub> とも 2-3 ショートです。

図 4 の SW<sub>1</sub> に関しては図 5 のようにセットします。

ディップ・スイッチ SW<sub>1</sub> の BANK(1~3)スイッチではバンク・メモリ・エリアのデコード・アドレスを設定します。バンク・メモリは使用しないので設定の必要はありません。

ディップ・スイッチ SW<sub>1</sub> の PORT(1~5)スイッチでは  $\mu$ PD72111 と I/O レジスタのデコード・アドレスを設定します。今回は  $\mu$ PD72111 のレジスタ・アドレス

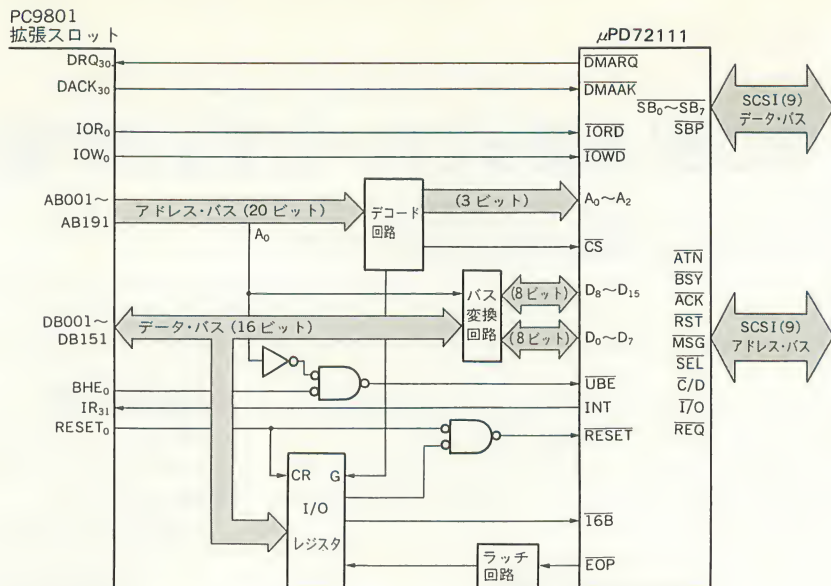


〈写真 1〉 SCSI コントロール・ボード

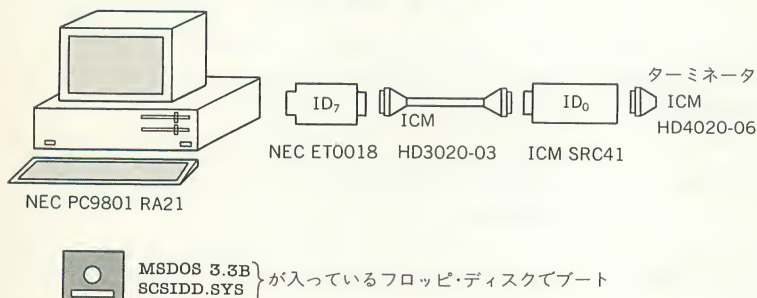


〈写真 2〉 SRC-41

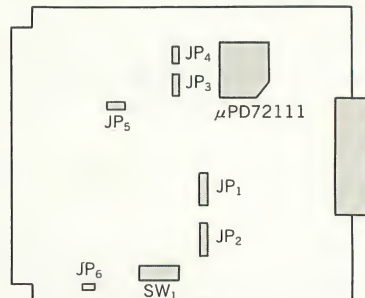
〈図 1〉  
SCSI コントロール・  
ボードのブロック図



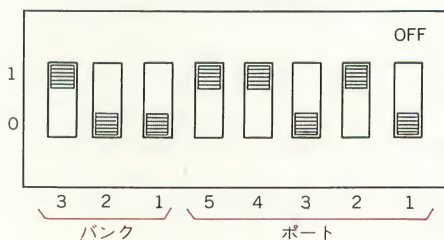
〈図 3〉 実験装置の接続



〈図 4〉 ET0018 スイッチ配置



〈図 5〉 SW<sub>1</sub>の設定



は 00D0H~00D7H, I/O レジスタ・アドレスは 01D0H にセットします。

なお, ET0018 の詳しい設定に関しては μPD72111 (SCSIC) の実習テキストを参考にして下さい。

## ソフトウェア構成

MS-DOS システムでハードディスクを使用するためのソフトウェア構成を図 6 に示します。

MS-DOS システムとは別に必要なソフトウェアはハードウェア依存部分をまとめた BIOS (図 6 の①) とディスクを MS-DOS で使用可能にするための論理フォーマット・プログラム (図 6 の②) および MS-DOS がブロック・デバイスとしてハードディスクを使用するためのデバイス・ドライバ (図 6 の③) の三つです。

以下にこの三つのプログラムの概要を述べ、詳細を次項以降で説明します。

### ● BIOS

BIOS はハードディスクの I/O サービスを行うハードウェア依存部分をまとめたプログラムです。

この BIOS をコールすることでハードウェアにとらわれず、論理的なレベルでハードディスクの I/O アクセスが行えます。

SCSI コマンド・ディスクリプタ・ブロック (CDB) を持って BIOS コールを行うことにより SCSI コマンドを実行します。

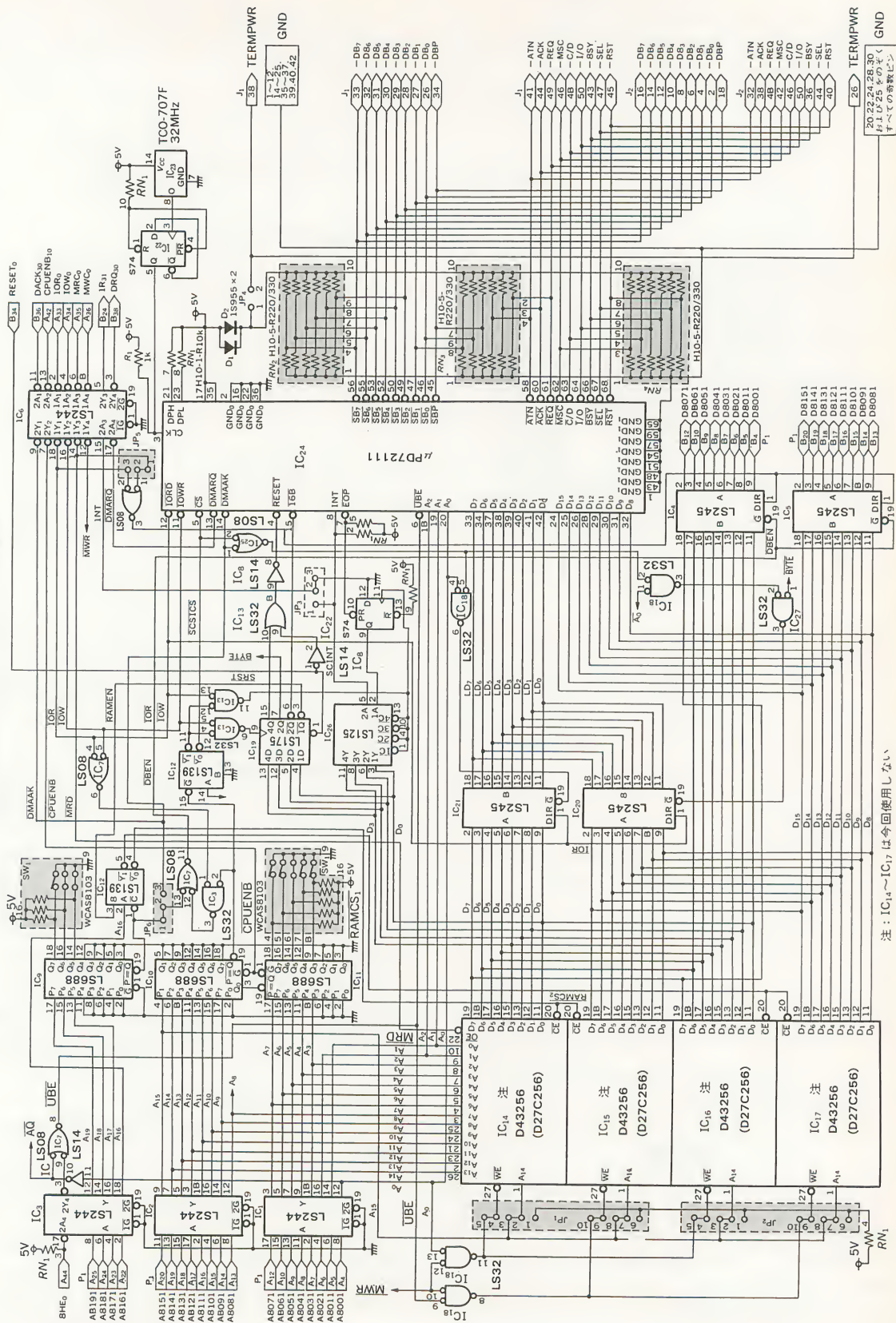
ハードディスクに対する BIOS コールは不特定多数のプログラムから利用できるようにソフトウェア割り込み (INT40H) で行うようにしました。

### ● 論理フォーマット・プログラム

論理フォーマット・プログラムはハードディスクに

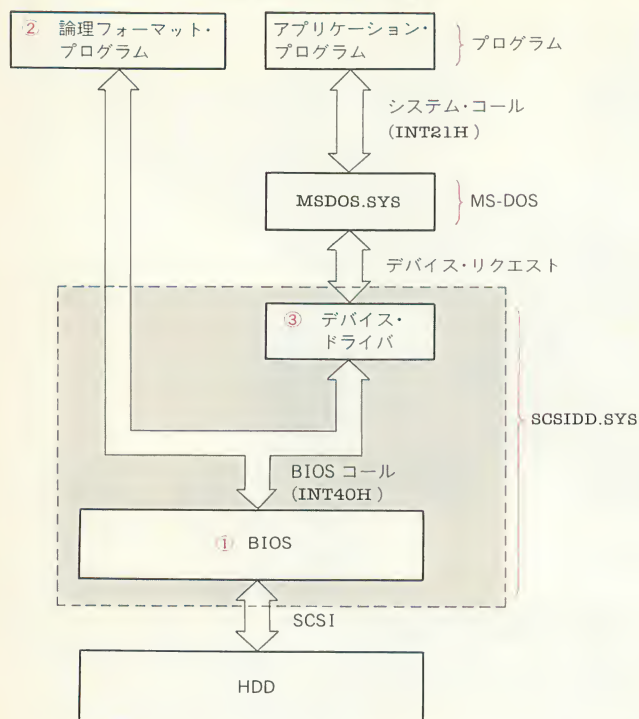


〈図2〉 SCSIコントローラ・ボードの回路図



注：IC<sub>14</sub>～IC<sub>17</sub>は今回使用しない

〈図6〉ソフトウェア構成



MS-DOSのブロック・デバイスとして必要な初期情報を書き込みます。通常、SCSI ハードディスクは物理フォーマットをした状態で出荷されています。

したがって、セクタ単位でアクセスすることは可能ですが、MS-DOSでブロック・デバイスとしてアクセスするにはあらかじめ論理フォーマットしておく必要があります。

#### ●デバイス・ドライバ

デバイス・ドライバはMS-DOSからハードディスクがブロック・デバイスとしてアクセスできるようにするプログラムです。

システム立ち上げ時に組み込まれ、ハードディスクをファイル・デバイスとして使用可能にします。

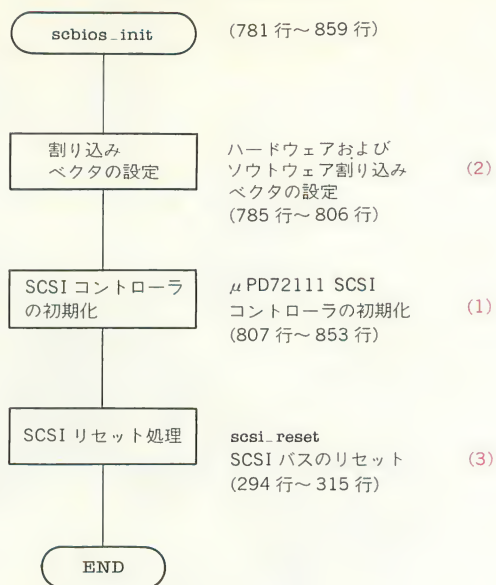
デバイス・ドライバはBIOSコールを行うので、BIOSもデバイス・ドライバと同時にしくはそれ以前にロードされていることが必要です。今回の例ではデバイス・ドライバとBIOSをひとつのプログラムとして一括でロードしています。

### BIOS

SCSI デバイスをアクセスするハードウェア依存部分をBIOSとしてまとめています。

このBIOSコールを不特定多数のプログラムから利用できるようにソフトウェア割り込み(INT40H)で行うようにしました。BIOSのプログラム・リストを

〈図7〉イニシャライズ



リスト1に示します。また、プログラム中の各処理のフローチャートを図7〜図13に示します。

以降の各項で示される行番号はリスト1の行番号です。

#### ●イニシャライズ

BIOSを使用する前にscbios\_init(781行~895行)をコールして各部をイニシャライズすることが必要です。図7にフローチャートを示します。

(1) SCSI コントローラのハードウェア割り込みとBIOS エントリのソフトウェア割り込みのベクタを設定

ハードウェア割り込みのマスクも解除します。

(2)  $\mu$ PD72111 SCSI コントローラを初期化

モードの設定、セレクション・タイム・アウトとREQ/ACK タイム・アウトの設定、自分のIDの設定を行い、コントローラがアイドル状態に入るのを待ちます。

(3) SCSI リセット処理

リセット・コマンドを実行し、終了を待ちます。

#### ●BIOS エントリ

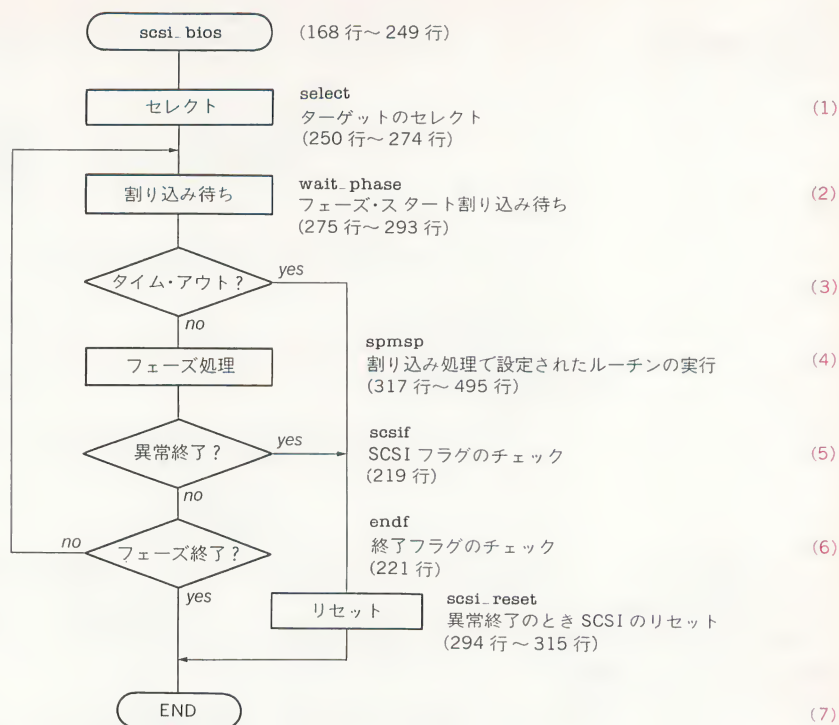
SCSI BIOSの実行はパラメータをレジスタに設定したのちにソフトウェア割り込みを実行することで行います。結果はALおよびAHレジスタにセットされます。

#### ▶入力パラメータ

DS: BX CDB(コマンド・ディスクリプ



〈図8〉メイン・ルーチン



DX                    タ・ブロック)のアドレス  
ES : DI              CDB のバイト数  
CX                    データ・バッファ・アドレス  
                      データのバイト数

▶ 終了パラメータ

AL                    SCSI ステータス・バイト  
                      (133 行)  
AH                    SCSI フラグ  
                      (147 行~156 行)

● BIOS ソフトウェア割り込み処理

ソフトウェア割り込み処理ルーチン(メイン・ルーチン)のフローチャートを図8に示します。メイン・ルーチンはハードウェア割り込みによって起動され、各フェーズ処理を行います。

(1) ターゲットをセレクト

SCSI コントローラのアイドル状態を待ちます。アイドル状態になると BIOS ハードウェア割り込みを許可し、ターゲット ID をセット、そしてセレクト・コマンドを発行、タイム・アウト・カウンタをセットし、割り込み発生を待ちます。

(2) フェーズ・スタート割り込みを待つ

(3) タイム・アウトであれば SCSI リセット処理を行い、タイム・アウト・フラグを終了パラメータにセットして終了

(4) 各フェーズの処理

フェーズ・スタート割り込みがあると SCSI フェーズ処理スタート・ポインタに必要な処理ルーチンのポ

インタがセットされるので、そのフェーズ処理を実行します。SCSI フェーズ処理スタート・ポインタのセットは次の項で述べる BIOS ハードウェア割り込みで行います。

フェーズ処理はコマンド・フェーズ処理、データ・フェーズ処理、ステータス・フェーズ処理、メッセージ・イン・フェーズ処理があります。これらについては後述します。

(5) 実行されたフェーズの終了状態をチェックし、異常終了であれば SCSI リセット

SCSI リセット処理は SCSI コントローラのアイドル状態を待ち、リセット・コマンドを発行します。

(6) フェーズ終了でなければ、フェーズ・スタート割り込み待ち(2)に戻る

(7) フェーズ終了ならば BIOS 処理を終了

終了処理をして BIOS ハードウェア割り込みを禁止し、SCSI ステータス、SCSI フラグを AX レジスタに返り値としてセットします。

● BIOS ハードウェア割り込み処理

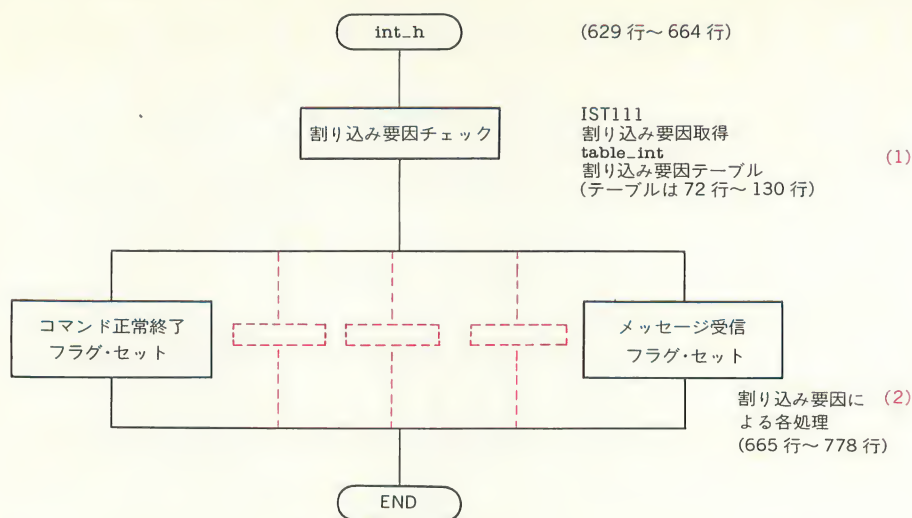
ハードウェア割り込み処理ルーチンを図9に示します。このルーチンはμPD72111 SCSI コントローラからの割り込みに対する処理を行います。

(1) SCSI コントローラからの割り込み要因を調べる

(2) 割り込み要因によって各処理ルーチンに入る

各処理ではメイン・ルーチンの処理を制御するフラグの設定、もしくはメイン・ルーチンで処理されるべきルーチン・ポインタの設定を行います。

〈図 9〉  
ハードウェア割り込み  
ルーチン



ここで使用されるフラグおよびポインタは 141 行 ~ 156 行に示される各パラメータです。

以下の各フェーズは SCSI のフェーズと対応します。

### ● コマンド・フェーズ処理

コマンド・フェーズ処理ルーチンを図 10 に示します。イニシエータからターゲットへ CDB(コマンド・ディスクリプタ・ブロック)データを転送します。

- (1) SCSI コントローラのアイドル状態を待つ
- (2) CDB のトランスファ・カウントを設定し、トランスファ・コマンドを発行
- (3) CDB の長さのデータを出力

REQ/ACK を見ながら CDB データを 1 バイトずつデータ FIFO レジスタに書き込み、ターゲットに転送します。

- (4) コマンドの正常終了を待つ

正常終了で戻ってこないときはタイム・アウト・エラーで終了します。

### ● データ・フェーズ処理

データ・フェーズ処理ルーチンを図 11 に示します。このルーチンはデータ・フェーズ・モードの設定にもとづいてデータ・イン・フェーズ処理もしくはデータ・アウト・フェーズ処理を実行します。データ・イン・フェーズ処理はターゲットからイニシエータへのデータ転送、データ・アウト・フェーズ処理はイニシエータからターゲットへのデータ転送です。

- (1) DMA の転送モードを設定

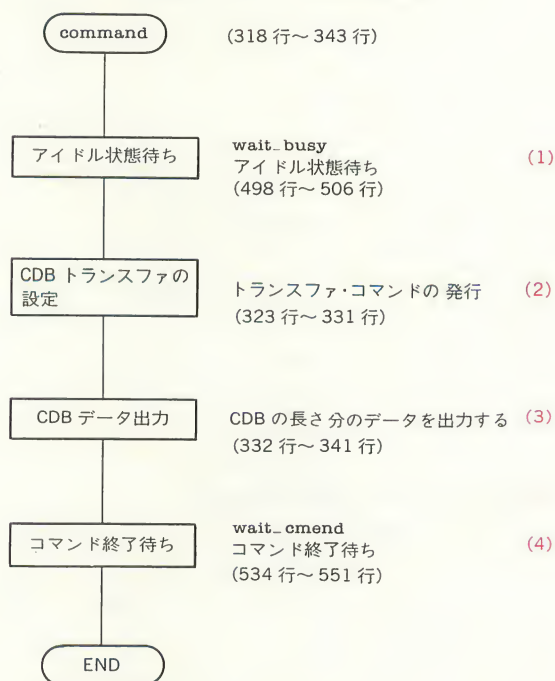
DMA のチャンネル 3 に対してシングル・モード、アドレス・インクリメント、オート・イニシャライズ禁止のモードと、ライト転送かリード転送かを設定します。

- (2) DMA コントローラを設定

DMA バンク・レジスタのセット、DMA ベース・アドレスのセット、DMA カウンタのセットを行います。

- (3) データの長さをトランスファ・カウンタに設定し、

〈図 10〉 コマンド・フェーズ処理



トランスファ・コマンドを発行

- (4) コマンドの終了を待つ

タイム・アウト・カウントをセットし、コマンドの正常終了を待ちます。正常終了で戻ってこない場合はタイム・アウト・エラーで終了します。

### ● ステータス・フェーズ処理

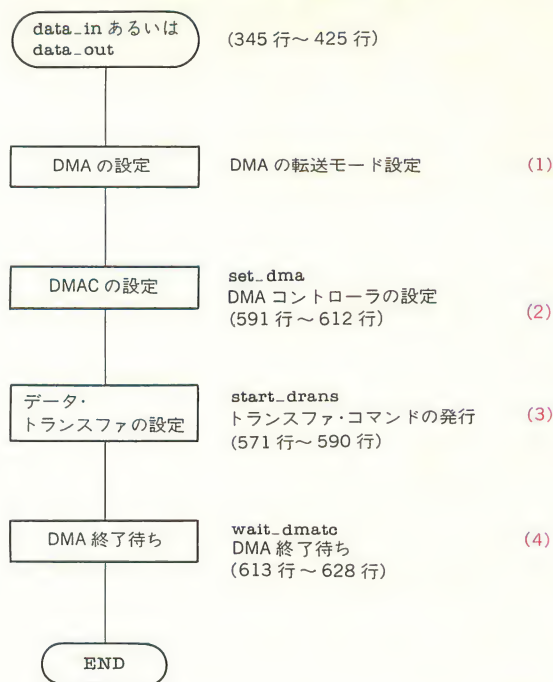
ステータス・フェーズ処理ルーチンを図 12 に示します。ターゲットからステータス・バイトを読みとります。

- (1) トランスファ・コマンドをスタート

- (2) ステータス・データを入力



〈図 11〉 データ・フェーズ処理



データ FIFO レジスタから 1 バイトをリードし、リードしたデータをステータス領域にセットします。設定時間内にステータスを受け取らなかった場合はタイム・アウト・エラーになります。

### (3) コマンドの正常終了を待つ

タイム・アウト・カウントをセットし、コマンドの正常終了を待ちます。正常終了で戻ってこない場合はタイム・アウト・エラーで終了します。

### ●メッセージ・イン・フェーズ処理

メッセージ・イン・フェーズ処理ルーチンを図 13 に示します。ターゲットからのメッセージを読みとります。

#### (1) トランスファ・コマンドをスタート

#### (2) ターゲットからメッセージ・データを入力

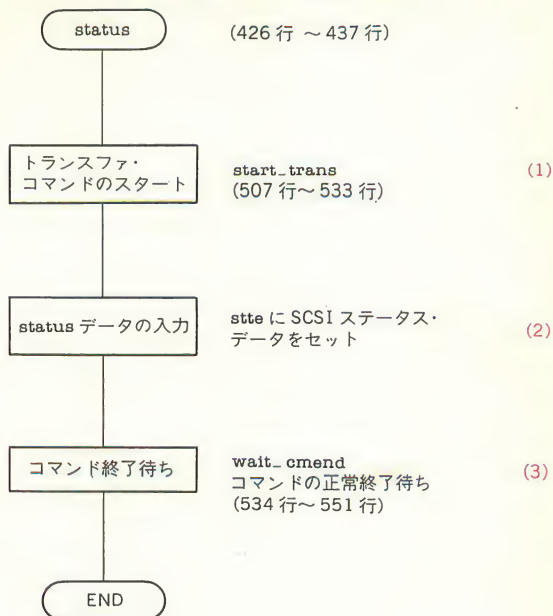
データを FIFO レジスタから 1 バイト・リードし、リードしたデータをメッセージ・イン領域にセットします。設定時間内にメッセージを受け取らなかった場合はタイム・アウト・エラーになります。

(3) 入力メッセージがコマンド・コンプリート・メッセージであればディスクコネクを待ち、その他のメッセージに対してはメッセージ・リジェクトを出力をするために A T N を出力

## 論理フォーマット

論理フォーマットとは MS-DOS のブロック・デバイスとして必要な初期情報をディスクに書き込むこと

〈図 12〉 ステータス・フェーズ処理



です。デバイス・ドライバを介してハードディスクを使用する前に論理フォーマットを行う必要があります。

論理フォーマット・プログラムの構成は C 言語で記述された部分 FORMATHD.C とアセンブラで記述された部分 BIOSHD.ASM からなります。C 言語部分 FORMATHD.C をリスト 2 に、アセンブラ部分 BIOSHD.ASM をリスト 3 に示します。以下で示す行番号はリスト 2 のものです。

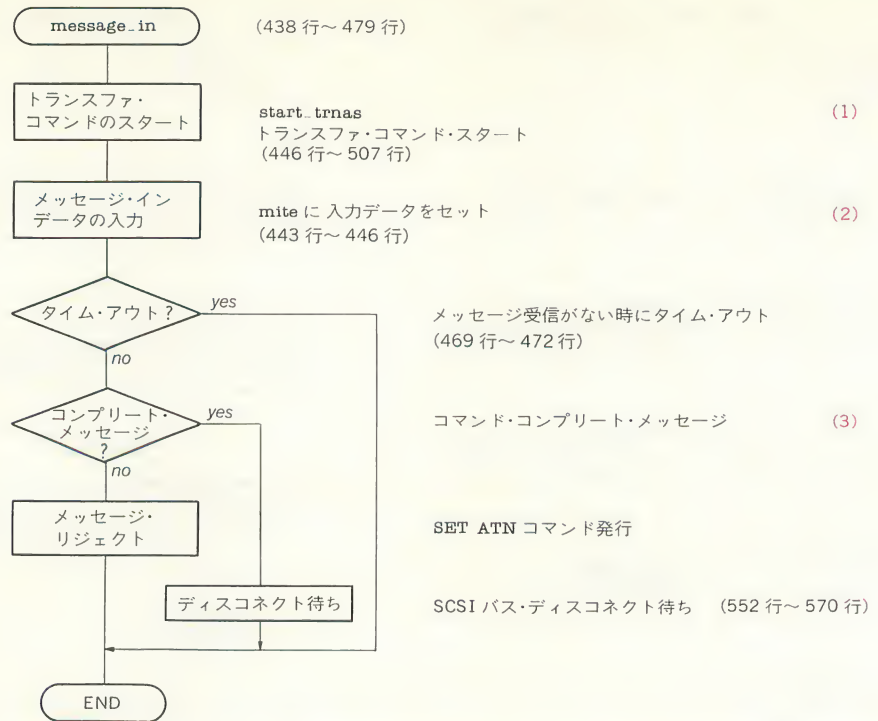
### ●ブロック・デバイスの初期情報

MS-DOS のブロック・デバイスはデバイス・ドライバを通して見ると固定サイズのブロックの配列になっています。この状態を図 14 に示します。ブロック配列は先頭から順に予約領域 (BIOS パラメータ・ブロック)、FAT 領域、ルート・ディレクトリ領域、ファイル領域の 4 領域に分けて使用されます。

ところで MS-DOS ver.3.X における論理セクタ番号は 16 ビットであつかわれるので論理セクタ番号の最大は 65535 です。したがって 512 バイト/物理セクタとすると、ブロック・デバイスの最大容量は 32M バイトです。そこで 40M バイトのディスクをひとつのブロック・デバイスとしてあつかうためには論理セクタのサイズが最低 1024 バイトである必要があります。ここではふたつの物理セクタ (512 バイト/セクタ) を合わせてひとつの論理セクタとして使用することで 64M バイトまでをひとつのブロック・デバイスとしてあつかえるようにしました。

FAT 領域とルート・ディレクトリ領域は初期化されていることが必要です。また、一般的に論理セクタ 0 (予約領域) に BPB (BIOS パラメータ・ブロック、後

〈図 13〉  
メッセージ・イン・  
フェーズ処理



〈図 14〉  
ブロック・デバイスの  
レイアウト

論理セクタ番号	領 域 名 称	初 期 設 定 デ ー タ
0	予約領域(BPB) 1セクタ(2セクタ)	
1 2 3 4	第1のFAT領域 4セクタ(8セクタ)	F8 FF FF 00 ~ 00
5 6 7 8	第2のFAT領域 (第1のFAT領域のコピー) 4セクタ(8セクタ)	F8 FF FF 00 ~ 00
9 ┆ 104	ルート・ディレクトリ領域 96セクタ(192セクタ)	00 00 ~ 00 00
105 ┆ 39599	ファイル領域	

- ・ 512バイト/セクタの物理セクタふたつをひとつの論理セクタとした場合の40Mバイト・ディスクのレイアウト例
- ・ セクタの( )は物理セクタ数

述)を設定します。以下、これらの論理セクタについて述べていきます。

予約領域は論理セクタ0から始まります。MS-DOSは必ずしも予約領域を必要とするわけではありません。しかし、BPB(BIOSパラメータ・ブロック)を論理セクタ0に入れておくことで任意長のディスクに対応できるため論理ブロック0を予約領域とし、この中にBPBを入れておくことが一般的です。

● BPB(予約領域内のパラメータ・ブロック)

BPBの構成を表1に示します。

BPBはMS-DOSが予約領域、FAT領域、ルート・ディレクトリ領域、ファイル領域の四つの領域の大きさを決めるためのパラメータです。以下でBPBの各パラメータについて説明します。なお、BPB内のパラメータで使うセクタとは論理セクタのことであり、ここでは物理セクタを二つ続けたものを論理セクタとしています。

(1) セクタ当たりのバイト数(論理セクタ・サイズ)

MS-DOSがディスク・アクセスするときのセクタ・サイズが何バイトであるかを設定します。

ここでは物理セクタ(512バイト)×2ですから、



〈表 1〉

BIOS パラメータ・ブロック  
(BPB)の構成

オフセット	パラメータ内容	バイト	設定値
00H	セクタ当たりのバイト数	2	1024
02H	アロケーション・ユニット(1 クラスタ)当たりのセクタ数(2 のべき乗でなければならない)	1	16
03H	予約セクタ数	2	1
05H	FAT 数	1	2
06H	ディレクトリ数	2	3072
08H	論理ボリュームの総セクタ数	2	39600
0AH	メディア・ディスクリプタ・バイト	1	F8H
0CH	FAT セクタ数	2	4

1024 バイトです。

(2) アロケーション・ユニット当たりのセクタ数

ファイルにディスクの領域を割り当てる単位です。

ここでは **16 論理セクタ**です。したがって **16K バイト**になります。この単位のことを **MS-DOS** では **クラスタ**と呼びます。

ファイル長が1バイトのファイルでもディスク上の領域はこの単位で割り当てられています。したがって1バイトから16Kバイトまでのファイルが占有するディスク領域はどれも同じ16Kバイトです。

(3) 予約セクタ数

予約領域としてディスクの先頭から何セクタを **MS-DOS** の **ファイル領域外**としてあつかうかを設定します。

ここではBPBのために1論理セクタを確保しています。また、ブート可能なデバイスの場合には一般的に**ブート・ストラップ・ロード**をBPBとともにこのセクタに入れておきます。

(4) FAT 数

FAT(ファイル・アロケーション・テーブル)のコピーの数です。

FATとはMS-DOSが**ディスク上のファイル**を**配置**、**管理するためのテーブル**です。したがってFATが読み出せないとファイルをまったくアクセスできなくなります。

そこで、第1のFATが壊れたときのために第1のFATのコピーである**第2のFAT**を用意します。通常フロッピー・ディスクやハード・ディスクの場合はFATをふたつ用意します。FATの内容はFAT領域の項で説明します。

(5) ディレクトリ数

ルート・ディレクトリの総エントリ数を設定します。

この値により、ルート・ディレクトリの大きさである**論理セクタ数**が決まります。計算方法は、

$$\left( \frac{\text{ディレクトリ・エントリ長(32バイト)} \times \text{ディレクトリ数}}{\text{論理セクタ・サイズ}} \right)$$

＝ルート・ディレクトリの**論理セクタ数**となります。ただし除算の端数は切り上げます。

(6) 論理ボリュームの総セクタ数

ディスク内の**ブロック・デバイス**として使用する**総**

**論理セクタ数**を設定します。

(7) メディア・ディスクリプタ・バイト

フロッピー・ディスクなどでメディアの種類を識別するために使用されます。

**ハードディスクの場合特に意味を持ちませんが**、一般的に**F8H**を使用するのでここでも**F8H**にしました。

(8) FAT セクタ数

ひとつのFATが占有するセクタ数を設定します。

FATの配列の要素数は**総クラスタ数**です。総クラスタ数は**論理ボリュームのセクタ数をクラスタ当たり**のセクタ数で割った値になります。この値にFATの要素のサイズ(この場合12ビット)を掛けた値がFATの**総ビット数**になります。このビット数をセクタまで切り上げた数がFATセクタ数です。

本実験での設定値の場合における計算式を示します。

$$\begin{aligned} 39600 &\div 16 = 2475 \\ \left( \begin{array}{l} \text{論理ボリューム} \\ \text{の総セクタ数} \end{array} \right) &\left( \begin{array}{l} \text{クラスタ当} \\ \text{りのセクタ数} \end{array} \right) & \left( \begin{array}{l} \text{総クラ} \\ \text{スタ数} \end{array} \right) \\ 2475 &\times 12 \text{ ビット} = 29700 \text{ ビット} \\ 29700 \text{ ビット} \div 8 \text{ ビット} \div 1024 &= 4 \text{ セクタ} \end{aligned}$$

(1セクタのバイト数)

なお、除算の端数は切り上げます。

● FAT 領域

FAT(ファイル・アロケーション・テーブル)領域はMS-DOSが**ディスク上のファイル配置をクラスタ単位で管理するための配列**です。図15に示すようにクラスタには2から順に番号がつけられます。この番号がFATの配列番号に対応します。

Ver.2.X以前のMS-DOSでは配列番号は12ビットで表されていました。したがって配列番号は12ビットで表せる数(0~4095)でした。

後述するようにMS-DOSでは0, 1および**F8FH**~**F8FFH**を特別な番号として使用しているので、実際に使用できる配列番号の数は4078です。したがってクラスタの最大数は4078です。

ディスクの容量が大きいとクラスタ数を4078以下にするためには、クラスタ当たりの大きさを32Kバイト, 64Kバイトなどとかなり大きくしなければなりません。

そこでVer. 3. X以後のMS-DOSでは12ビットの

他に 16 ビット (0~65535) で配列番号を表わすモードが追加されました。この場合も 0, 1 および FFF0H ~ FFFFH を特別な番号として使用しているので、実際に使用できる配列番号の数は 65518 です。ただし、FAT のサイズが大きくなるので MS-DOS のオーバ・ヘッドは重くなります。

今回の実験に使用したディスクの容量は 40 M バイトと比較的小容量なので 12 ビット FAT を使います。

以下でファイル・アクセスの場合を例にとり、簡単に FAT の使用方法を説明します。

アクセスするファイルに対応する **ディレクトリ・エントリ** (ファイル名とその他のパラメータにより構成される 32 バイトのデータ、後述) **の中にある開始クラスタ番号がそのファイルのデータ開始クラスタ** を指します。それと同時にこの番号は FAT 中のエントリ (FAT の配列の要素) の番号を指しており、**FAT 中のエントリの内容はそのファイルの次のクラスタ番号を保持しています。**

エントリの値が **FF8H~FFFH** の場合、このエントリが **ファイル中の最後のクラスタ** です。また、未使用のクラスタに当たるエントリには **000H** が設定されます。

この他にクラスタ番号以外の意味を持つものとして MS-DOS の **予約番号である FFOH~FF6H** までと **不良クラスタを示す FF7H** があります。

FAT の先頭のふたつのエントリはメディア・ディスクリプタ・コードである **FF8H** と **FFFH** が予約データとして設定されており、クラスタ 0, 1 は未使用になります。また、これに続く FAT 領域の初期値としてはその 3 バイトを除いて **000H** がセットされます。

### ●ルート・ディレクトリ領域

ルート・ディレクトリ領域は BPB のディレクトリ数で設定された数分のディレクトリ・エントリを持ったディレクトリのテーブルです。ひとつのディレクトリ・エントリは 32 バイトで構成され、**ファイル名、拡張子、属性、日付け、開始クラスタ番号、ファイル容量** の各パラメータで構成されています。これを図 16 に示します。

### ●データ領域

ディレクトリ領域の後ろにクラスタ 2 番の先頭論理セクタが配置されます (前述のとおりクラスタ 0 および 1 は FAT 中の予約コードで使用されるため、データ領域はクラスタ 2 番から始まる)。ここから論理ボリュームの総セクタ数で表される最終セクタまでがデータ領域であり、MS-DOS がディレクトリおよび

FAT によって管理し、ファイル領域として使用する領域です。

### ●論理フォーマット・プログラムの動作

論理フォーマット・プログラムはデバイス・ドライバ内の BIOS を使用しているので、デバイス・ドライバが MS-DOS に組み込まれた環境で実行します。

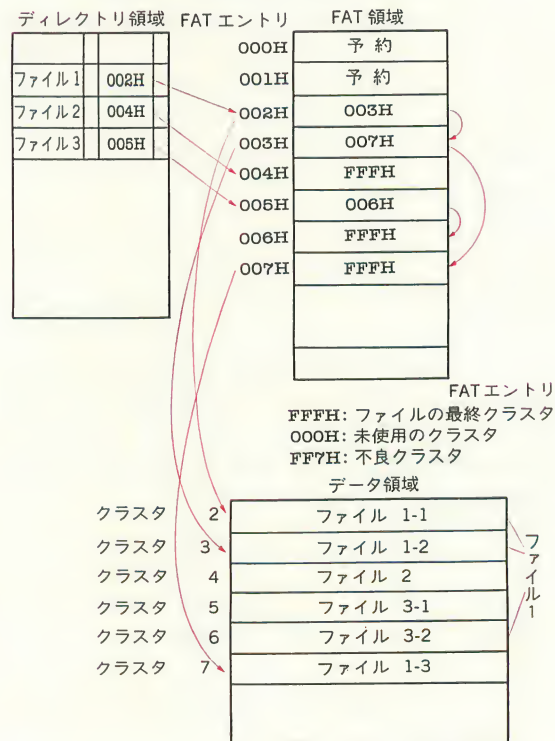
論理フォーマット・プログラムはディスク・ドライブの論理ブロック先頭番地から順に必要なデータを書き込んでいきます。

ICM SRC-41 以外のディスクを使用する場合には BPB (予約領域内のパラメータ・ブロック) の項で説明したように BPB の内容を変更する必要があります。FORMATHD.C (リスト 2) の 6 行~18 行で示されるように、BPB データはフォーマット・プログラム内定数として定義されています。変更する場所は 17 行目の定数です。

64M バイト以下のディスクの場合には **論理ボリュームの総セクタ数を容量に応じて変更し**、これをもとに **クラスタ当たりのセクタ数を変更して FAT セクタ数を計算します。**

ディスクの容量が 64M バイト以上の場合には論理

〈図 15〉 FAT とクラスタの関係



〈図 16〉 ディレクトリ・エントリ

00	~	07	08~0A	0B	0C~15	16~19	1A	1B	1C	~	1F
ファイル名			拡張子	属 性	予 約	日付け	開 始 クラスタ	ファイル容量			



セクタ・サイズを 2048 バイト以上にすることになります。ただし MS-DOS がブート・アップ時に IO.SYS のパラメータから決定する論理セクタ・サイズ以下でなければなりません。

FAT およびディレクトリの初期化データは BPB の内容によってフォーマット・プログラムが自動的に作るので変更の必要はありません。

論理フォーマット手順のフローチャートを図 17 に示します。

(1) プログラム中の定数をもとに BPB のデータを作成し、書き込む(make\_bpb()関数)

BPB 書き込み用バッファをクリアし、メーカー名、BPB データをセットして予約セクタに書き込みます。

(2) BPB の内容をもとに FAT データを作成し、書き込む(init\_fat()関数)

FAT 書き込み用バッファをクリアし、FAT 先頭の FAT-ID、予約コード、未使用コードで初期データを作り FAT 領域に書き込みます。

(3) BPB の内容をもとにルート・ディレクトリの初期データを作成し、FAT の後に書き込む(init\_dir()関数)

ルート・ディレクトリ書き込み用バッファをクリアし、BPB のディレクトリ数からディレクトリ・セクタ数を求めてルート・ディレクトリ領域に書き込みます。

以上で、ハードディスクをブロック・デバイスとして使用するために必要な初期情報の書き込みが終了します。MS-DOS でブロック・デバイスとして使用するにはこの後リセットして MS-DOS システムを再起動

する必要があります。

## デバイス・ドライバ

MS-DOS は図 6 に示したようにハードディスクをブロック・デバイスとしてアクセスするためのリクエストをデバイス・ドライバに出します。このリクエストを受け取ったデバイス・ドライバが BIOS を通じて実際にハードディスクをアクセスします。

デバイス・ドライバはアセンブラで記述された部分 SCSUB.ASM と C で記述された部分 SCDV.C および先に述べた BIOS 部分 SCBIOS.ASM から構成されます。アセンブラ部分 SCSUB.ASM をリスト 4 に、また C 言語部分 SCDV.C をリスト 5 に示します。

### ●デバイス・ヘッダ

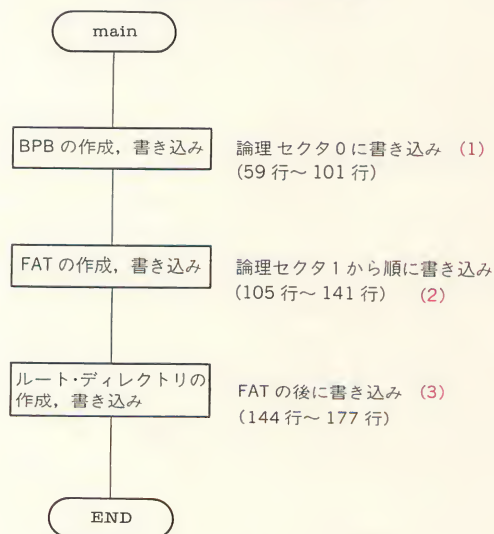
デバイス・ドライバは MS-DOS への管理情報として先頭にデバイス・ヘッダを持っています。その下にデバイス・ドライバのプログラムがあります。本プログラムのデバイス・ヘッダの値は表 2 のように設定されています。

次のデバイスへのリンク情報は次のデバイス・ヘッダへのポインタで示し、最後は -1 です。本プログラムではデバイス・ヘッダは一つなので -1 にします。

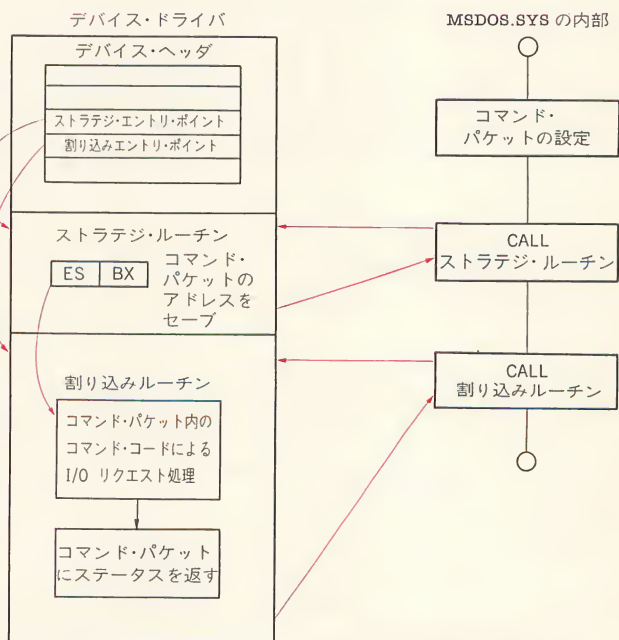
デバイスの属性はブロック型デバイス・ドライバです。ユニット数は 1 に設定しています。

ストラテジ・エントリ・ポイントと割り込みエントリ・ポイントはプログラムへのポインタです(図 18)。実際の動作については後述します。

〈図 17〉 フォーマット手順



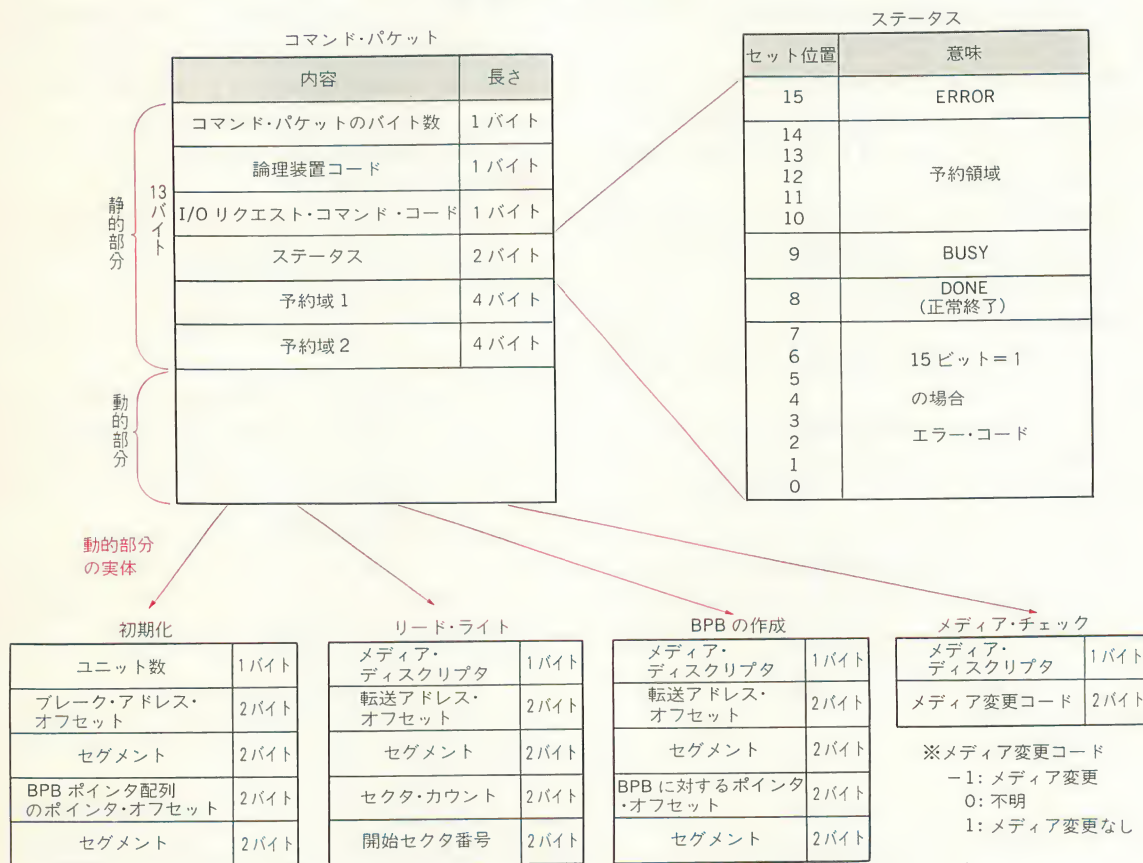
〈図 18〉 デバイス・ドライバのリクエスト



〈表2〉 デバイス・ヘッダ

	オフセット	内 容	バイト	設定値
デバイス・ヘッダ	0000H	次のデバイスへのリンク情報	4	-1
	0004H	デバイスの属性	2	2000H
	0006H	ストラテジ・エントリ・ポイント	2	strategy_r
	0008H	割り込みエントリ・ポイント	2	interrupt_r
	000AH	デバイス名またはユニット数	8	1
デバイス本体	0012H ↓ XXXXH	プログラム	可変長	

〈図19〉 コマンド・パケット



ストラテジ・エントリ・ポイントは**ストラテジ・ルーチン**のアドレスを格納しています。ストラテジ・ルーチンは**MS-DOS からのコマンド・パケットのアドレス**(セグメント: オフセット)を受け取るためのものです。

割り込みエントリ・ポイントは割り込みルーチンのアドレスを格納しています。割り込みルーチンは MS-DOS からのコマンド・パケットによって指示された I/O リクエスト処理を実際に行います。

### ● ストラテジ・ルーチン

MS-DOS はデバイス・ドライバに対して処理を行いたいとき、まず始めにコマンド・パケット(図19)を設定した後、そのポインタを ES: BX に格納してストラテジ・ルーチンを呼び出します。

次に割り込みルーチンを呼び出して処理を行います。ストラテジ・ルーチンはこのポインタを後で実行される割り込みルーチンのためにセーブしておきます。

このように二段階で処理を行うのはマルチプロセス OS で一般的に行われる手法で、**マルチプロセス OS との互換性**を考えて二段階の処理ステップを取っているわけです。

### ● 割り込みルーチン(I/O リクエスト処理)

割り込みルーチンの中では各 I/O リクエストが処理されます。

I/O リクエスト処理はコマンド・パケット中のコマンド・コードにより、表3に示す I/O リクエスト処理に分岐して実行されます。

以下の各項で示している行番号はリスト5の行番



〈表3〉 I/O リクエスト・コードと処理関数

コード	関 数 名	処 理
00H	init()	初期化
01H	media_check()	メディア・チェック
02H	build_bpb()	BPBの作成
04H	input()	入力
08H	output()	出力
09H	output_v()	ペリファイ付き出力
その他	error_io()	エラー・ステータスの設定

号です。

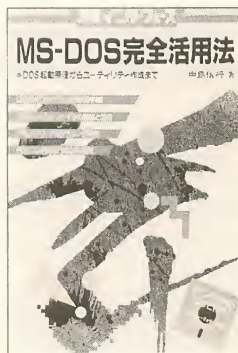
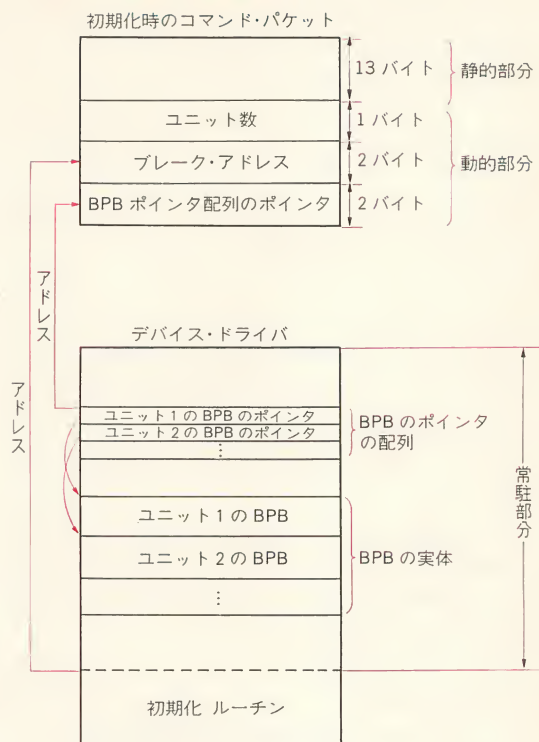
(1) 初期化(init())関数, 360行~419行)

初期化はI/O イニシャライズ処理, テスト・レディ処理を実行し, コマンド・パケットにユニット数として1を, 加えてブレーク・アドレス(prog\_seg, prog\_off)とBPB(表1を参照のこと)ポインタ配列のポインタを設定します。この関係を図20に示します。

I/O イニシャライズ処理はSCSI BIOSの初期化, デバイス・ドライバを組み込んだことの表示を行います。

テスト・レディ処理(611行~688行)はハードディスクがレディ状態であるかどうかSCSI BIOSでテスト・ユニット・レディ・コマンドを送信して確認します。もしレディ状態でなければSCSI BIOSでスター

〈図20〉 初期化時のコマンド・パケット



# MS-DOS完全活用法

●DOS起動原理からユーティリティ作成まで 中島信行 著

B5判, 224頁, 2色刷, 5"2HD FD付き, 定価2,500円(税込), 送料260円

本書では, まず, MS-DOSやプログラムがどのようなメカニズムで起動するかを探ります。技術向上を目指すプログラマにとって, プログラムの動作原理を知ることは重要です。つぎに, MS-DOSの機能を十分に活用するために, 各システム・ファイルの機能を解析し, アセンブラやC言語を使って, 機能活用のユーティリティを多数作成します。さらに, パソコンが用意するROM BIOSについて, その利用法を研究します。5インチ2HDフロッピー付き(全70ファイル)。



● 本書のおもな内容 ●

第一部 MS-DOS動作メカニズム

- 第1章 リセットからMS-DOSが起動するまで
- 第2章 プログラムの起動原理
- 第3章 簡単なプログラムの作成
- 第二部 MS-DOSシステム・ファイルの機能と活用
- 第4章 IO.SYSの役割とデバイス・ドライバの構造
- 第5章 MSDOS.SYSの役割とCTRLキー制御機能
- 第6章 COMMAND.COMの役割と標準入出力

第三部 アセンブリ・プログラミング手法

- 第7章 アセンブリ・プログラミングの実例
- C言語とのインターフェース関数の作成 ——
- 第8章 C言語とアセンブリ言語とのベンチマーク・テスト
- 第四部 パソコンROM BIOSの活用研究
- 第9章 PC-9801シリーズのRS-232-C ROM BIOSの活用
- 第10章 IBM PCのRS-232-C ROM BIOSの差し替え
- 第11章 RS-232-C ROM BIOSの活用
- 第12章 デバック手法

CQ出版社

〒170 東京都豊島区巣鴨1-14-2 出版部 ☎(03)5395-2121 振替 東京0-10665  
営業部 ☎(03)5395-2141

ト・ストップ・コマンドを送信し、もう一度レディかどうかを確認します。

それでもエラーが発生した場合にはコマンド・パケットにエラーをセットします。

ユニット数はデバイス・ドライバが何台のドライブをサポートするのかを示します。ここでは1台なので1です。

**BPBの配列へのポインタはユニット数分のBPBに対するポインタの配列を指し示します。**

ブレイク・アドレスはデバイス・ドライバ・プログラムの常駐部分の終了アドレスを示します。

初期化ルーチンは始めに一度呼び出されるだけで初期化ルーチンを常駐部分より後に置いて解放してしまうのが一般的です。

(2) メディア・チェック処理(media\_check())関数, 422行~434行]

ブロック型デバイス・ドライバでメディアが交換できる物の場合には**メディア交換時にBPBの再作成やFATの読み直しをします**。そのためにメディアが交換されたかどうかを調べます。

メディア・チェック時のコマンド・パケットの動的部分は図19に示すようにメディア・ディスクリプタとメディア変更コードです。メディア変更コードはメディアが変更された場合-1, 不明の場合0, メディアが変更されない場合には1です。

このドライバ場合, メディアは固定なので常に変更されていない場合の値である1をコマンド・パケットのメディア変更コードに設定し, ステータスにDONEビットをセットします。

(3) BPBの作成(build\_bpb())関数, 437行~454行]

**MS-DOSはメディアにアクセスする前にメディアのBPBの作成を要求します。**

BPB作成時のコマンド・パケットの動的部分はメディア・ディスクリプタ, 転送アドレス, BPBへのポインタです(図19)。転送アドレスは1セクタ分のバッファのポインタで, 作業領域として使うことができます。ただし, 本プログラムでは使用していません。

ここでは論理フォーマット・プログラムによってハードディスクの先頭セクタ(セクタ番号0)に書かれたBPBデータを作業領域に読み込み, BPBへのポインタ, BPBへのメディア・ディスクリプタ・バイトをコマンド・パケットにセットします。

(4) 入力処理(input())関数, 471行~492行]

論理セクタ単位でデータの読み込みを行います。

コマンド・パケットの動的部分からデータのバッフ

ァ先頭アドレス, 読み込む論理セクタ数, 読み込み開始論理セクタ番号を取り出します。これらをBIOSパラメータに設定し, リード・エクステンド・コマンドをscsi\_read()関数で実行してハード・ディスクからデータを読み出します。

コマンドの終了状態をコマンド・パケットのステータスにセットします。正常終了であればDONEビットをセットし, 異常終了であればエラー・ステータスをセットします。

(5) 出力処理(output())関数, 495行~515行]

論理セクタ単位でデータの書き込みを行います。

コマンド・パケットの動的部分からデータのバッファ先頭アドレス, 書き込み論理セクタ数, 書き込み開始論理セクタ番号を取り出します。これらをBIOSパラメータに設定し, ライト・エクステンド・コマンドをscsi\_write()関数で実行してディスクにデータを書き込みます。

コマンドの終了状態をコマンド・パケットのステータスにセットします。正常終了であればDONEビットをセットし, 異常終了であればエラー・ステータスをセットします。

(6) ベリファイ付き出力処理(output\_v())関数, 518行~538行]

論理セクタ単位でデータのベリファイ書き込みを行います。

scsi\_write()関数の代わりにscsi\_writenv()関数を実行して書き込みと同時にベリファイを行うことを除けば出力処理と同じです。

(7) エラー処理(error\_io())関数, 457行~468行]

(1)から(6)以外の処理はコマンドが認識できないのでコマンド・パケットのステータスにI/Oリクエスト・コマンドが間違いであるというエラーをセットし, 終了します。

MS-DOSから要求されたこれらの処理をBIOSを通してハードディスクに実行してやることによってMS-DOSのファイル・システムとしてハードディスクを使用することが可能になります。

#### ◆引用\*・参考文献◆

- (1)\*NEC,  $\mu$ PD72111 SCSI コントローラ・アプリケーションノート。
- (2)\*NEC,  $\mu$ PD72111(SCSIC)実習テキスト。
- (3) 特集 MS-DOS とデバイスドライバ, トラ技コンピュータ, 1989. 11, pp54-91, CQ 出版社。
- (4) MS-DOS エンサイクロペディア Volume 1, pp.517-561, アスキー。
- (5)\*NEC,  $\mu$ PD72111 ユーザーズマニュアル。



1:	SCSI BIOS FOR #PD72111																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						</
----	------------------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

＜リスト1＞ SCBIOS.ASM(つづき)

[illegible]

```

238:      ah,scsif
239:      bp
240:      pop ds
241:      pop ds
242:      pop di
243:      pop si
244:      pop dx
245:      pop bx
246:      pop bx
247:      iret
248:
249:      ;
250:      ; scsi_bios      endp
251:
252:      ; セレクト ターゲッド
253:      ; output エラーコードに依る異常終了
254:      ; cflag
255:      select:
256:      255:      call      wait_busy
257:      256:      mov     al,00h
258:      259:      mov     dx,01b_111
259:      258:      out     dx,al
260:      259:      out
261:      260:      al,18h
262:      260:      mov     dx,01b_111
263:      260:      out     dx,al
264:      263:      selectw:
265:      264:      cmp     scsif,00h
266:      265:      jnc     selecte
267:      266:      cmp     cnf,01h
268:      267:      jnc     selectw
269:      268:      mov     cnf,00h
270:      269:      mov     motc,MSG_IDEN
271:      270:      ret
271:
272:      ;
273:      ; Selecte:
274:      272:      stc
273:      273:      ret
274:
275:      ; フェーズ・スタート・割り込み発生待ち
276:      ; output タイムアウト この場合タイムアウト・フラグをセット
277:      ; cflag
278:      275:      wait_phase:
279:      280:      mov     toc,0200h
281:      280:      mov     toc,0FFFFh
282:      281:      mov     wait_phase:
283:      283:      cmp     phsf,01h
284:      284:      jnc     wait_phaseg
285:      285:      dec     wait_phaseg
286:      286:      jnc     wait_phasew
287:      287:      dec     toc2
288:      288:      jnc     wait_phasew
289:      289:      or      wait_phasew
290:      290:      scsif,00h
291:      290:      stc
292:      291:      wait_phase:
293:      292:      ret
293:
294:      ; SCS Iリセット
295:      ; output
296:      294:      ; cflag タイムアウト
297:      297:      scsi_reset:
298:      298:      call      wait_busy
299:      299:      300:      al,00h
300:      300:      mov     dx,01b_111
301:      301:      mov     al,00h
302:      302:      out     dx,al
303:      303:      mov     al,cmdend
304:      304:      out     dx,al
305:      305:      jnc     scsi_resete
306:      306:      mov     cx,10
307:      307:      scsi_resetw:
308:      308:      push     cx
309:      309:      xor     loop,S
310:      310:      loop     scsi_resetw
311:      311:      pop     cx
312:      312:      cld
313:      313:      scsi_resete:
314:      314:      315:      ret
315:
316:      ;

```





```

475: mov al,04h
476: mov dx,CMD_111
477: out dx,al
478: call wait_discon
479: jmp msg_1_end
480:
481: ; メッセージ・アウト・フェーズ処理
482:
483: message out:
484: start_trans
485: msg_o_end
486: je
487: mov dx,CMD_111
488: out dx,al
489: call wait_cmencl
490: je
491: test scsif,20h
492: je
493: call wait_discon
494: msg_o_end:
495: ret
496:
497: subttl フェーズ処理のサブルーチン
498:
499: ; アイドル状態を待つ
500:
501: wait_busy:
502: mov dx,CST_111
503: in al,dx
504: test al,80h
505: jne wait_busy
506: ret
507:
508: ; TRANSFER コマンドスタート
509: output
510: cf: on REQ / ACK タイムアウト・エラー
511:
512: start_trans:
513: call
514: al,002h
515: mov dx,CMD_111
516: out dx,al
517:
518: ; DRQ = 1 を待つ
519: output
520: cf: on REQ / ACK タイムアウト・エラー
521:
522: wait_drq:
523: cnp scsif,00h
524: jne wait_drq
525: mov dx,CST_111
526: in al,dx
527: test al,01h
528: je wait_drq
529: ret
530:
531: wait_drq:
532: stc
533: ret
534:
535: ; コマンド正常終了待ち
536: output
537: cf: on タイムアウト この場合タイムアウト・フラグをセット
538:
539: wait_cmencl:
540: mov cx,0
541: wait_cmencl:
542: cnp chsf,01h
543: loopnz wait_cmencl
544: jne wait_cmencl
545: mov chsf,00h
546: ret
547:
548: wait_cmencl:
549: or
550: stc
551: ret
552:
553: ; ディスコネクト待ち

```

```

554: ; wait_discon:
555: mov cx,0
556: wait_discon:
557: scsif,80h
558: test
559: loopnz wait_discon
560: je
561: and
562: scsif,7fh
563: mov al,05h
564: mov dx,CMD_111
565: out dx,al
566: mov
567: endf,01h
568: ret
569:
570: ; タイムアウト
571: wait_discon:
572: scsif,00h
573: or
574: ret
575:
576: ; データ転送コマンドスタート
577:
578: start_dtrans:
579: call
580: b,dcs_length
581: mov al,BTCL_111+80h
582: mov dx,ADR_111
583: out dx,al
584: mov al,bl
585: mov dx,WINI_111
586: out dx,al
587: mov al,bh
588: mov dx,WINI_111
589: out dx,al
590: mov al,00h
591: mov dx,al
592: out dx,al
593: mov al,12h
594: mov dx,CMD_111
595: out dx,al
596: ret
597:
598: ; DMA の設定
599: input
600: al: バンクアドレス
601: bx: 64 K バンクのオフセット
602: cx: count value
603:
604: set_dmac:
605: out
606: out
607: mov ax,bx
608: mov al,ah
609: mov al,ah
610: mov ax,cx
611: dec ax
612: out
613: out
614: mov al,03h
615: mov al,03h
616: out
617: ret
618:
619: ; DMA 終了待ち
620: output
621: cf: on REQ / ACK タイムアウト・エラー
622:
623: wait_dmac:
624: cnp
625: jne wait_dmac
626: in
627: test al,08h
628: je wait_dmac
629: ret
630:
631: ; wait_dmac:
632: stc
633: ret
634:
635: ; REQ / ACK タイムアウト・エラー
636: subttl ハードウェア割り込み処理
637:
638: ; REQ / ACK タイムアウト・エラー

```



リスト1 SCBIOS.ASM(つづき)

634: ; μPD72111からの割り込み要因を解析し、各処理へ分岐する。

```

634: int_h
635: proc
636: push ax
637: push bx
638: push cx
639: push dx
640: push si
641: push di
642: push bp
643: mov ax, cs
644: mov ds, ax
645: mov dx, 10h
646: in al, dx
647: mov bx, offset MGROUP; table, int_3
648: mov bp, 3
649: add bp, 3
650: mov al, [bx]
651: jmp int_table
652: call word ptr [bx+1]
653: mov al, 63h
654: out port, al
655: ds
656: pop di
657: pop si
658: pop dx
659: pop cx
660: pop bx
661: pop ax
662: iret
663:
664: int_h
665: endp
666:
667: ; コマンド正常終了処理
668:
669: cnef, 01h
670: mov
671:
672: ; F I F O オーバーラン、アングラン・エラー処理
673:
674: fow
675:
676: or scsif, 01h
677: ret
678:
679: ; セレクト・タイムアウト・エラー処理
680:
681: stoe
682:
683: or scsif, 08h
684: ret
685:
686: ; REQ/ACK タイムアウト・エラー処理
687:
688: r1aoc
689:
690: or scsif, 02h
691: ret
692:
693: ; REQ/ACK タイムアウト・エラー
694:
695: f1aoc
696:
697: or scsif, 04h
698: ret
699:
700: ; SCS I リセット・コンディション処理
701:
702: src
703:
704: or scsif, 40h
705: ret
706:
707: ; ディスコネクト処理
708:
709: disc
710:
711: or scsif, 80h
712: ret
713:
714: ; ディスコネクトフラッグ
715:
716: mess_r
717:
718: mov mrf, 01h
719:
720: ; メッセージ受信フラッグ

```

```

719: ret
720:
721: ; リセット処理
722:
723: rstf, 01h
724: mov
725: ret
726:
727: ; リセットフラッグ
728:
729: scsif, 01h
730: mov
731: ret
732:
733: ; SCS I バス・パリティ・エラー処理
734:
735: or scsif, 10h
736: mov
737: ret
738:
739: ; ホスト・バス・パリティ・エラー処理
740:
741: scsif, 10h
742: mov
743: ret
744:
745: ; データ・アウト・フェーズ・スタート処理
746:
747: s p m s p にデータ・フェーズ処理のポインタをセット
748:
749: mov spmsp_offset, _TEXT: data_out
750: mov phsf, 01h
751: ret
752:
753: ; データ・イン・フェーズ・スタート処理
754:
755: s p m s p にデータ・フェーズ処理のポインタをセット
756:
757: mov spmsp_offset, _TEXT: data_in
758: mov phsf, 01h
759: ret
760:
761: ; コマンド・フェーズ・スタート処理
762:
763: s p m s p にコマンド・フェーズ処理のポインタをセット
764:
765: mov spmsp_offset, _TEXT: command
766: mov phsf, 01h
767: ret
768:
769: ; ステータス・フェーズ・スタート処理
770:
771: s p m s p にステータス・フェーズ処理のポインタをセット
772:
773: mov spmsp_offset, _TEXT: status
774: mov phsf, 01h
775: ret
776:
777: ; メッセージ・イン・フェーズ・スタート処理
778:
779: s p m s p にメッセージ・イン・フェーズ処理のポインタをセット
780:
781: mov spmsp_offset, _TEXT: message_in
782: mov phsf, 01h
783: ret
784:
785: ; メッセージ・アウト・フェーズ・スタート処理
786:
787: s p m s p にメッセージ・アウト・フェーズ処理のポインタをセット
788:
789: mov spmsp_offset, _TEXT: message_out
790: mov phsf, 01h
791: ret
792:
793: ; 物理 I/O 初期化ルーチン
794:
795: subttl 物理 I/O 初期化
796:
797: scbios_init
798:
799: mov dx, 10h
800: out dx, al
801:
802: ; 8ビット・モードに設定
803:
804: cli
805:
806: ; CPU への割り込み要求待ち

```

# リスト 1> SCBIOS.ASM (つづき)

```

791: mov dx, CST_111
792: in al, dx
793: test al, 40h
794: je test
795: mov dx, INT_111
796: mov dx, 13_111
797: mov ax, 0
798: mov es, ax
799: mov es:INT_VEC_OFF, offset TEXT:INT_H ;ハード割り込みベクタセット
800: mov es:INT_VEC_SEG, es
801: mov es:INT_BIOSVEC_OFF, offset TEXT:sest_bios ;int 40h (100h, 102h)
802: mov es:INT_BIOSVEC_SEG, es
803: in al, 0F7h
804: out 0F7h, al
805: cull INT_55, al
806: sti
807:
808: mov al, MOD_111
809: mov dx, ADDR_111
810: out dx, al
811: mov al, 000h
812: out dx, al
813:
814:
815: mov dx, WIN_111
816: out dx, al
817:
818: mov al, TMOAD_111
819: mov dx, ADDR_111
820: out dx, al
821: mov al, 00h
822: out dx, al
823: mov dx, WIN_111
824: out dx, al
825:
826: mov al, SRTOUT_111
827: out dx, al
828: mov al, 10h
829: out dx, al
830: mov dx, WIN_111
831: out dx, al
832:
833: mov al, RATOUT_111
834: out dx, al
835: mov al, 000h
836: out dx, al
837: mov dx, WIN_111
838: out dx, al
839:
840: mov al, PID_111
841: out dx, al
842: mov al, 87h
843: out dx, al
844:
845:
846: mov cneq, 00h
847: mov al, 00h
848: call wait_busy
849: mov al, 00h
850: mov dx, DID_111
851: out dx, al
852: call scsi_reset
853: mov al, 80h
854: out dx, al
855: mov dx, DID_111
856: out dx, al
857: scbios_init
858:
859:
860: _TEXT ENDS
861:
862:
863:
864: END

```

# リスト 2> FORMATHD.C

```

1: /* # P D 7 2 1 1 フォーマット (FORMATHD, C) */
2:
3: #include <stdio.h>
4:
5: /* BPB の構造体 */
6: struct bpb_st {
7:     /* 1 セクタ当りのバイト数 */
8:     int b_sector;
9:     /* 1 セクタ当りのセクタ数 */
10:    char sector_c;
11:    /* 予約セクタ */
12:    int resv_sector;
13:    /* FAT の数 */
14:    int fat;
15:    /* ディレクトリ数 */
16:    int dir;
17:    /* 論理ボリームの総セクタ数 */
18:    char med_id;
19:    /* マデイト・ディスクリプタ・バイト */
20:    int fat_sec;
21:    /* FAT セクタ数 */
22:    int bpb_h = {
23:        1024, 16, 1, 2, 3072, 39600, 0xf8, 4
24:    };
25:
26:    /* Jump Code & Maker Name Data */
27:    char *jump_data[11] = {
28:        "C", "M", "D", "I", "S", "C"
29:    };
30:
31:    /* Define P_SECTOR_SIZE 512 */
32:    /* Define BPB Size 13 */
33:    /* Define Jump & Maker Data Size 11 */
34:    /* Define Success Flag 0 */
35:
36:    /* Logical Start Sector Number */
37:    long stno;
38:    /* Logical Sector Length (Counts) */
39:    int sec_len;
40:    /* Work Buffer (1024 Bytes) */
41:    char buf[1024];
42:    /* HD Status */
43:    int hstat;
44:    /* BPB Data Pointer */
45:    char *bpb_ptr;
46:    /* Logical Sector / Physical Sector */
47:    int l_per_p;
48:
49:    main()
50:    {
51:        l_per_p = bpb_h.b_sector / P_SECTOR_SIZE;
52:
53:        if (make_bpb() != OK) { /* BPB の作成、書き込み */
54:            printf(stderr, "Y007Can not Make bpb. Yn.");
55:        }
56:
57:        if (init_fat() != OK) { /* FAT の初期化、書き込み */
58:            printf(stderr, "Y007Can not Initialize FAT. Yn.");
59:        }
60:
61:        if (init_dir() != OK) { /* ルートディレクトリの初期化、書き込み */
62:            printf(stderr, "Y007Can not Initialize Directory. Yn.");
63:        }
64:
65:        /* Make BPB by bpb_h */
66:        /* Return Value: OK = 0, Error = Not Zero */
67:        make_bpb()
68:        i, n, m;
69:
70:        hstat = OK;
71:        bpb_ptr = char *bpb_h;
72:        for (i = 0; i < bpb_h.b_sector; ++i) {
73:            buf[i] = 0; /* 書き込みセクタデータのクリア */
74:        }
75:
76:        stno = 0;
77:        sec_len = 1;
78:
79:        for (i = 0; i < bpb_h.resv_sector; ++i) {
80:            if (i == 0) { /* 1st Reserve Sector Data */
81:                n = 0; /* セクタ 0 のデータセット */
82:            }
83:        }
84:    }

```





## リスト 2 FORMATHD.C(つづき)

```

235: # Return Value:
236: # OK = 0      Error = Not Zero
237: #
238: #
239: #
240: #
241: #
242: #
243: #
244: #
245: #
246: #
247: #
248: #
249: #
250: #
251: #
252: #
253: #
254: #
255: #
256: #
257: #
258: #
259: #
260: #
261: #
262: #
263: #
264: #
265: #
266: #
267: #
268: #
269: #
270: #
271: #
272: #
273: #
274: #
275: #
276: #
277: #
278: #
279: #
280: #

```

## リスト 3 BIOSHD.ASM

```

1: ;
2: ; PAGE 132,56
3: ; _TEXT SEGMENT WORD PUBLIC
4: ; ASSUME CS:_TEXT
5: ;
6: ;
7: ;
8: ;
9: ;
10: ;
11: ;
12: ;
13: ;
14: ;
15: ;
16: ;
17: ;
18: ;
19: ;
20: ;
21: ;
22: ;
23: ;
24: ;
25: ;
26: ;
27: ;
28: ;
29: ;
30: ;
31: ;
32: ;
33: ;
34: ;
35: ;
36: ;
37: ;
38: ;
39: ;
40: ;
41: ;
42: ;
43: ;
44: ;
45: ;
46: ;

```

## リスト 4 SCSUB.ASM

```

1: ; デバイス・ドライバ
2: ; アセンブラ部分
3: ;
4: ;
5: ;
6: ;
7: ;
8: ;
9: ;
10: ;
11: ;
12: ;
13: ;
14: ;
15: ;
16: ;
17: ;
18: ;
19: ;
20: ;
21: ;
22: ;
23: ;

```



```

47: prog_off dw ? ;プログラム終了アドレスのセーブ領域
48: prog_seg dw ?
49:
50: hyouji db 'SCSIデバイス・ドライバ Ver 1.1を組み込みました。'
51: 0dh,0ah,0dh,0ah,'$'
52:
53: work_buf db 1024*3 dup(?) ;Work Buffer 3→3
54: label byte ;スタック先頭アドレス
55: DATA ENDS
56:
57: PEND SEGMENT WORD PUBLIC 'PEND'
58: progend label word ;プログラム終了アドレス
59: PEND ENDS
60:
61: public _packet
62: _packet offset _prog_seg ;コマンド・パケットのポインタ領域
63: public _work_buf ;プログラム終了アドレスのセーブ領域
64: public _site ;IPB 読み込みバンプ
65: public _scsi ;ステータスコード
66: ;SCSIリザルフラグ
67:
68: TEXT SEGMENT BYTE PUBLIC 'CODE'
69: ***** ;SCSIリザルフラグの領域 *****
70: *****
71: *****
72: header label word ;次のデバイスへのポインタ
73: dd -1 ;デバイスの属性
74: dw 2000h ;ストラテジ・エントリ・ポインタ
75: dw strategy_r ;ストラテジ・エントリ・ポインタ
76: dw interrupt_r ;割り込みエントリ・ポインタ
77: dd 1 ;エニット数
78: *****
79: ***** ;コマンド・パケットの置かれているアドレスを格納する。 出力: packet *****
80: ***** ;ストラテジ・ルーチン *****
81: *****
82: *****
83: *****
84: *****
85: strategy_r proc far
86: mov word ptr cs: packet, bx
87: mov word ptr cs: packet+2, es
88: ret
89:
90:
91:
92: strategy_r endp
93: *****
94: *****
95: *****
96: ***** ;SCSIデバイス・ドライバの入口と出口。この処理の 入力: なし 出力: _prog_off, _prog_seg *****
97: ***** ;IPでエントリ処理をCALLする。 *****
98: *****
99: *****
100: interrupt_r proc far
101:
102: push ax ;レジスタ退避
103: push bx
104: push cx
105: push dx
106: push si
107: push di
108: push bp
109: push ds
110: push es
111:
112: mov ax, cs
113: mov ds, ax
114: mov word ptr stacksave, sp
115: mov word ptr stacksave+2, ss
116:
117: cli
118: mov ss, ax
119: mov sp, offset DGROUP:stacktop
120: sti
121:
122: mov prog_off, offset DGROUP:progend ;プログラムの終了アドレスを_prog_seg_prog_offにセット
123: mov _prog_seg, cs
124: call _d_entry ;エントリ処理
125:

```

```

126: cli
127: mov sp, word ptr stacksave
128: mov ss, word ptr stacksave+2
129: sti
130:
131:
132: pop es ;スタック・ポインタの復帰
133: pop ds
134: pop bp
135: pop si
136: pop dx
137: pop cx
138: pop bx
139: pop ax
140: ret
141:
142: interrupt_r endp
143: ***** ;SCSIリザルフラグの領域 *****
144: *****
145: ***** ;SCSIデバイス・ドライバを組み込みました。"と表示する。 入力: なし 出力: なし *****
146: ***** ;割り込みエントリのポインタ *****
147: ***** ;SCSIリザルフラグ *****
148: *****
149: *****
150: *****
151: *****
152: io_init proc near
153: bp
154: push si
155: push di
156: call ds: offset DGROUP:hyouji
157: mov ah, 09h
158: int 21h
159:
160: pop di
161: pop si
162: pop bp
163: ret
164: io_init endp
165: *****
166: ***** ;c から、物理B I/Oをコールするためのサブルーチン *****
167: *****
168: *****
169: scsi_bios proc
170: di
171: push es
172:
173: mov bx, cta
174: mov dh, 0
175: mov cx, 1
176: mov cx, ctdc
177: mov di, dha off
178: mov es, dha_seg
179: int 40h
180: mov _site, al
181: mov _scsi, ah
182: mov al, ah
183: pop es
184: pop di
185: ret
186: scsi_bios endp
187:
188:
189:
190: ***** ;MS-C Routine *****
191: public _actused
192: _actused
193:
194: proc
195: _actused
196: endp
197:
198:
199:
200:
201:
202:
203: END

```

```

1: 2: /* # P D 7 2 1 1 1 デバイス、ドライバ (S C D V, C) */
3: 3:
4: 4: /* #include <dos.h> */
5: 5:
6: 6:
7: 7:
8: /* コマンド・パケット・ステータス情報 */
9: 8:
10: 9: #define ST_ERROR 0x8000 /* ERROR ビット */
11: 10: #define ST_DONE 0x1000 /* DONE ビット */
12: 11: #define ST_ERR_NOTRDY 2 /* ドライブの準備ができていない */
13: 12: #define ST_ERR_BADCMD 3 /* コマンド番号が間違っている */
14: 13: #define ST_ERR_CRC 4 /* CRC エラー */
15: 14: #define ST_ERR_SECTOR 5 /* セクタが壊れている */
16: 15: #define ST_ERR_WRITE 8 /* 書き込みがうまくいけません */
17: 16: #define ST_ERR_READ 10 /* 読みのエラー */
18: 17: #define ST_ERR_GENERAL 12 /* その他のエラー */
19: 18:
20: /* センサ・コード情報 */
21: 19:
22: 20: #define R_R 0x17 /* リード・ヘッド・リニアリティ・エラー */
23: 21: #define EDC 0x18 /* EDC エラー */
24: 22: #define N_R 0x14 /* リード・ヘッド・リニアリティ・エラー */
25: 23: #define N_SF 0x15 /* リード・ヘッド・リニアリティ・エラー */
26: 24: #define N_SF 0x02 /* リード・ヘッド・リニアリティ・エラー */
27: 25: #define SE 0x15 /* シフト・エラー */
28: 26: #define W_F 0x03 /* リード・ヘッド・リニアリティ・エラー */
29: 27: #define N_TOF 0x06 /* リード・ヘッド・リニアリティ・エラー */
30: 28: #define P_DF 0x02 /* リード・ヘッド・リニアリティ・エラー */
31: 29: #define P_O 0x29 /* リード・ヘッド・リニアリティ・エラー */
32: 30: #define N_R 0x04 /* リード・ヘッド・リニアリティ・エラー */
33: 31:
34: /* コマンド、データのバイト数 */
35: 32:
36: 33: #define RSD_BC 0x06 /* リード・ヘッド・リニアリティ・エラー */
37: 34: #define HSD_BC 0x06 /* リード・ヘッド・リニアリティ・エラー */
38: 35: #define REC_BC 0x16 /* リード・ヘッド・リニアリティ・エラー */
39: 36: #define REC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
40: 37: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
41: 38: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
42: 39: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
43: 40: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
44: 41: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
45: 42: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
46: 43: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
47: 44: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
48: 45: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
49: 46: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
50: 47: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
51: 48: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
52: 49: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
53: 50: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
54: 51: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
55: 52: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
56: 53: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
57: 54: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
58: 55: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
59: 56: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
60: 57: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
61: 58: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
62: 59: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
63: 60: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
64: 61: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
65: 62: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
66: 63: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
67: 64: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
68: 65: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
69: 66: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
70: 67: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
71: 68: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
72: 69: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
73: 70: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
74: 71: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
75: 72: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
76: 73: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
77: 74: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
78: 75: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
79: 76: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
80: 77: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
81: 78: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
82: 79: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
83: 80: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
84: 81: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
85: 82: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
86: 83: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
87: 84: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
88: 85: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
89: 86: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
90: 87: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
91: 88: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
92: 89: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
93: 90: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
94: 91: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
95: 92: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
96: 93: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
97: 94: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
98: 95: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
99: 96: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */
100: 97: #define WEC_BC 0x0A /* リード・ヘッド・リニアリティ・エラー */

```

```

80: /* メディア・チェッカのコマンド・パケット */
81:
82:
83: struct {
84:     char media_cmd; /* DDPへのリクエストのタイプ */
85:     int retr; /* 返す値 */
86: } mc;
87:
88: /* インジヤライズのコマンド・パケット */
89:
90: struct {
91:     char unit; /* ユニットの数 */
92:     char tend_off; /* プレーク・アドレス */
93:     char tend_seg; /*
94:     struct bpb_st **bpb_dli_off; /* BPPの配列ポインタ */
95:     char bpb_dli_seg;
96:     } in;
97:
98: } up;
99:
100: command_packet;
101:
102: /* BIOS コマンド・パケット */
103:
104: typedef struct {
105:     int cmd; /* コマンド */
106:     char sector_count; /* セクタ カウント */
107:     char sector_count_s; /*
108:     char *transw_off; /* 転送アドレス */
109:     char *transw_seg; /*
110:     char start_sector_l; /* 開始セクタ番号 */
111:     char start_sector_h; /*
112:     char start_sector_m; /*
113:     char start_sector_lm; /*
114:     } cmd_bios_packet;
115:
116: /* BPPの構造体 */
117:
118: struct bpb_st {
119:     int b_sector; /* 1セクタ当りのバイト数 */
120:     char sector_c; /* 全セクタ数 */
121:     int resv_sector; /* 予約セクタ数 */
122:     char *fat; /* FATの表 */
123:     int fat_no; /* データの長さ */
124:     int tot_sec; /* 全セクタ数 */
125:     char med_id; /* メディア・ディスクリプタ・バイト */
126:     int fat_sec; /* FATセクタ数 */
127:     } bpb_h = {
128:         16,
129:         1,
130:         1,
131:         31,
132:         2,
133:         3072,
134:         39500,
135:         0xF8,
136:         4
137:     };
138:
139: /* エクステンディッド・コマンドの構造体 */
140:
141: struct extended_command {
142:     char ocr[2]; /* 開始セクタ番号 (High-MSB) */
143:     char start_sector; /* 開始セクタ番号 (MSB) */
144:     char start_sector_h; /* 開始セクタ番号 (High-LSB) */
145:     char start_sector_l; /* 開始セクタ番号 (LSB) */
146:     char fix_l; /* 転送セクタ数 (MSB) */
147:     char trans_sector_m; /* 転送セクタ数 (M) */
148:     char trans_sector_l; /* 転送セクタ数 (LSB) */
149:     char fix_h; /*
150:
151:     }
152:
153: /* リード・エクステンディッド・コマンド */
154:
155: char_rec = {
156:     0x28, 0x00, 0x00, 0x00,
157:     0x00,
158:     0x00,
159:     0x00,
160:     0x00,
161:     0x00,
162:     0x00,
163:     0x00,
164:     0x00,
165:     0x00,
166:     0x00,
167:     0x00,
168:     0x00,
169:     0x00,
170:     0x00,
171:     0x00,
172:     0x00,
173:     0x00,
174:     0x00,
175:     0x00,
176:     0x00,
177:     0x00,
178:     0x00,
179:     0x00,
180:     0x00,
181:     0x00,
182:     0x00,
183:     0x00,
184:     0x00,
185:     0x00,
186:     0x00,
187:     0x00,
188:     0x00,
189:     0x00,
190:     0x00,
191:     0x00,
192:     0x00,
193:     0x00,
194:     0x00,
195:     0x00,
196:     0x00,
197:     0x00,
198:     0x00,
199:     0x00,
200:     0x00,
201:     0x00,
202:     0x00,
203:     0x00,
204:     0x00,
205:     0x00,
206:     0x00,
207:     0x00,
208:     0x00,
209:     0x00,
210:     0x00,
211:     0x00,
212:     0x00,
213:     0x00,
214:     0x00,
215:     0x00,
216:     0x00,
217:     0x00,
218:     0x00,
219:     0x00,
220:     0x00,
221:     0x00,
222:     0x00,
223:     0x00,
224:     0x00,
225:     0x00,
226:     0x00,
227:     0x00,
228:     0x00,
229:     0x00,
230:     0x00,
231:     0x00,
232:     0x00,
233:     0x00,
234:     0x00,
235:     0x00,
236:     0x00,
237:     0x00,
238:     0x00,
239:     0x00,
240:     0x00,
241:     0x00,
242:     0x00,
243:     0x00,
244:     0x00,
245:     0x00,
246:     0x00,
247:     0x00,
248:     0x00,
249:     0x00,
250:     0x00,
251:     0x00,
252:     0x00,
253:     0x00,
254:     0x00,
255:     0x00,
256:     0x00,
257:     0x00,
258:     0x00,
259:     0x00,
260:     0x00,
261:     0x00,
262:     0x00,
263:     0x00,
264:     0x00,
265:     0x00,
266:     0x00,
267:     0x00,
268:     0x00,
269:     0x00,
270:     0x00,
271:     0x00,
272:     0x00,
273:     0x00,
274:     0x00,
275:     0x00,
276:     0x00,
277:     0x00,
278:     0x00,
279:     0x00,
280:     0x00,
281:     0x00,
282:     0x00,
283:     0x00,
284:     0x00,
285:     0x00,
286:     0x00,
287:     0x00,
288:     0x00,
289:     0x00,
290:     0x00,
291:     0x00,
292:     0x00,
293:     0x00,
294:     0x00,
295:     0x00,
296:     0x00,
297:     0x00,
298:     0x00,
299:     0x00,
300:     0x00,
301:     0x00,
302:     0x00,
303:     0x00,
304:     0x00,
305:     0x00,
306:     0x00,
307:     0x00,
308:     0x00,
309:     0x00,
310:     0x00,
311:     0x00,
312:     0x00,
313:     0x00,
314:     0x00,
315:     0x00,
316:     0x00,
317:     0x00,
318:     0x00,
319:     0x00,
320:     0x00,
321:     0x00,
322:     0x00,
323:     0x00,
324:     0x00,
325:     0x00,
326:     0x00,
327:     0x00,
328:     0x00,
329:     0x00,
330:     0x00,
331:     0x00,
332:     0x00,
333:     0x00,
334:     0x00,
335:     0x00,
336:     0x00,
337:     0x00,
338:     0x00,
339:     0x00,
340:     0x00,
341:     0x00,
342:     0x00,
343:     0x00,
344:     0x00,
345:     0x00,
346:     0x00,
347:     0x00,
348:     0x00,
349:     0x00,
350:     0x00,
351:     0x00,
352:     0x00,
353:     0x00,
354:     0x00,
355:     0x00,
356:     0x00,
357:     0x00,
358:     0x00,
359:     0x00,
360:     0x00,
361:     0x00,
362:     0x00,
363:     0x00,
364:     0x00,
365:     0x00,
366:     0x00,
367:     0x00,
368:     0x00,
369:     0x00,
370:     0x00,
371:     0x00,
372:     0x00,
373:     0x00,
374:     0x00,
375:     0x00,
376:     0x00,
377:     0x00,
378:     0x00,
379:     0x00,
380:     0x00,
381:     0x00,
382:     0x00,
383:     0x00,
384:     0x00,
385:     0x00,
386:     0x00,
387:     0x00,
388:     0x00,
389:     0x00,
390:     0x00,
391:     0x00,
392:     0x00,
393:     0x00,
394:     0x00,
395:     0x00,
396:     0x00,
397:     0x00,
398:     0x00,
399:     0x00,
400:     0x00,
401:     0x00,
402:     0x00,
403:     0x00,
404:     0x00,
405:     0x00,
406:     0x00,
407:     0x00,
408:     0x00,
409:     0x00,
410:     0x00,
411:     0x00,
412:     0x00,
413:     0x00,
414:     0x00,
415:     0x00,
416:     0x00,
417:     0x00,
418:     0x00,
4
```



```

159:   0x00,
160:   0x00,
161:   0x00,
162:   },
163: /* ライト・エクステンディッド・コマンド */
164:   cha_wcc = {
165:     0x7F,
166:     0x2A, 0x00, 0x00, 0x00,
167:     0x00,
168:     0x00,
169:     0x00,
170:     0x00,
171:     0x00,
172:     0x00,
173:     0x00,
174:     0x00,
175:     0x00,
176:     0x00,
177:     0x00,
178:     0x00,
179:     0x00,
180:     0x00,
181:     0x00,
182:     0x00,
183:     0x00,
184:     0x00,
185:     0x00,
186:     0x00,
187:     0x00,
188:     0x00,
189:     0x00,
190:     0x00,
191:     0x00,
192:     0x00,
193:     0x00,
194:     0x00,
195:     0x00,
196:     0x00,
197:     0x00,
198:     0x00,
199:     0x00,
200:     0x00,
201:     0x00,
202:     0x00,
203:     0x00,
204:     0x00,
205:     0x00,
206:     0x00,
207:     0x00,
208:     0x00,
209:     0x00,
210:     0x00,
211:     0x00,
212:     0x00,
213:     0x00,
214:     0x00,
215:     0x00,
216:     0x00,
217:     0x00,
218:     0x00,
219:     0x00,
220:     0x00,
221:     0x00,
222:     0x00,
223:     0x00,
224:     0x00,
225:     0x00,
226:     0x00,
227:     0x00,
228:     0x00,
229:     0x00,
230:     0x00,
231:     0x00,
232:     0x00,
233:     0x00,
234:     0x00,
235:     0x00,
236:     0x00,
237:     0x00,
238:     0x00,
239:     0x00,
240:     0x00,
241:     0x00,
242:     0x00,
243:     0x00,
244:     0x00,
245:     0x00,
246:     0x00,
247:     0x00,
248:     0x00,
249:     0x00,
250:     0x00,
251:     0x00,
252:     0x00,
253:     0x00,
254:     0x00,
255:     0x00,
256:     0x00,
257:     0x00,
258:     0x00,
259:     0x00,
260:     0x00,
261:     0x00,
262:     0x00,
263:     0x00,
264:     0x00,
265:     0x00,
266:     0x00,
267:     0x00,
268:     0x00,
269:     0x00,
270:     0x00,
271:     0x00,
272:     0x00,
273:     0x00,
274:     0x00,
275:     0x00,
276:     0x00,
277:     0x00,
278:     0x00,
279:     0x00,
280:     0x00,
281:     0x00,
282:     0x00,
283:     0x00,
284:     0x00,
285:     0x00,
286:     0x00,
287:     0x00,
288:     0x00,
289:     0x00,
290:     0x00,
291:     0x00,
292:     0x00,
293:     0x00,
294:     0x00,
295:     0x00,
296:     0x00,
297:     0x00,
298:     0x00,
299:     0x00,
300:     0x00,
301:     0x00,
302:     0x00,
303:     0x00,
304:     0x00,
305:     0x00,
306:     0x00,
307:     0x00,
308:     0x00,
309:     0x00,
310:     0x00,
311:     0x00,
312:     0x00,
313:     0x00,
314:     0x00,
315:     0x00,
316:     0x00,
317:     0x00,
318:     0x00,
319:     0x00,
320:     0x00,
321:     0x00,
322:     0x00,
323:     0x00,
324:     0x00,
325:     0x00,
326:     0x00,
327:     0x00,
328:     0x00,
329:     0x00,
330:     0x00,
331:     0x00,
332:     0x00,
333:     0x00,
334:     0x00,
335:     0x00,
336:     0x00,
337:     0x00,
338:     0x00,
339:     0x00,
340:     0x00,
341:     0x00,
342:     0x00,
343:     0x00,
344:     0x00,
345:     0x00,
346:     0x00,
347:     0x00,
348:     0x00,
349:     0x00,
350:     0x00,
351:     0x00,
352:     0x00,
353:     0x00,
354:     0x00,
355:     0x00,
356:     0x00,
357:     0x00,
358:     0x00,
359:     0x00,
360:     0x00,
361:     0x00,
362:     0x00,
363:     0x00,
364:     0x00,
365:     0x00,
366:     0x00,
367:     0x00,
368:     0x00,
369:     0x00,
370:     0x00,
371:     0x00,
372:     0x00,
373:     0x00,
374:     0x00,
375:     0x00,
376:     0x00,
377:     0x00,
378:     0x00,
379:     0x00,
380:     0x00,
381:     0x00,
382:     0x00,
383:     0x00,
384:     0x00,
385:     0x00,
386:     0x00,
387:     0x00,
388:     0x00,
389:     0x00,
390:     0x00,
391:     0x00,
392:     0x00,
393:     0x00,
394:     0x00,
395:     0x00,
396:     0x00,
397:     0x00,
398:     0x00,
399:     0x00,
400:     0x00,
401:     0x00,
402:     0x00,
403:     0x00,
404:     0x00,
405:     0x00,
406:     0x00,
407:     0x00,
408:     0x00,
409:     0x00,
410:     0x00,
411:     0x00,
412:     0x00,
413:     0x00,
414:     0x00,
415:     0x00,
416:     0x00,
417:     0x00,
418:     0x00,
419:     0x00,
420:     0x00,
421:     0x00,
422:     0x00,
423:     0x00,
424:     0x00,
425:     0x00,
426:     0x00,
427:     0x00,
428:     0x00,
429:     0x00,
430:     0x00,
431:     0x00,
432:     0x00,
433:     0x00,
434:     0x00,
435:     0x00,
436:     0x00,
437:     0x00,
438:     0x00,
439:     0x00,
440:     0x00,
441:     0x00,
442:     0x00,
443:     0x00,
444:     0x00,
445:     0x00,
446:     0x00,
447:     0x00,
448:     0x00,
449:     0x00,
450:     0x00,
451:     0x00,
452:     0x00,
453:     0x00,
454:     0x00,
455:     0x00,
456:     0x00,
457:     0x00,
458:     0x00,
459:     0x00,
460:     0x00,
461:     0x00,
462:     0x00,
463:     0x00,
464:     0x00,
465:     0x00,
466:     0x00,
467:     0x00,
468:     0x00,
469:     0x00,
470:     0x00,
471:     0x00,
472:     0x00,
473:     0x00,
474:     0x00,
475:     0x00,
476:     0x00,
477:     0x00,
478:     0x00,
479:     0x00,
480:     0x00,
481:     0x00,
482:     0x00,
483:     0x00,
484:     0x00,
485:     0x00,
486:     0x00,
487:     0x00,
488:     0x00,
489:     0x00,
490:     0x00,
491:     0x00,
492:     0x00,
493:     0x00,
494:     0x00,
495:     0x00,
496:     0x00,
497:     0x00,
498:     0
```

[illegible]

```

317: void bios_bpb_rec( ); /* Set BPB BIOS Read Command */
318: /**/
319: void init( ), media_check( ), build_bpb( ), input( ),
320: output( ), output_v( ), error_io( );
321:
322: /* 1/O リクエスト・テーブル */
323:
324: void (*func[10])( ) = {
325:     0 /* イニシャライズ処理
326:     1 /* メディア・チェック処理
327:     2 /* BUILD BPB 処理
328:     3 /* 1/O リクエスト・エラー処理
329:     4 /* インフット処理
330:     5 /* エラー処理
331:     6 /* 1/O リクエスト・エラー処理
332:     7 /* 1/O リクエスト・エラー処理
333:     8 /* アウトプット処理
334:     9 /* アウトプット・ウィズ・ベリファイ処理
335:     };
336:
337:
338:
339:
340:
341: /* エントリ処理
342:
343: /* 1/O リクエスト・コマンド・コードによって処理をCALLする。 入力: packet->cmd 出力: packet->command_status
344: /* 1/O リクエスト・コードが0AH以下のものが入力してくると、MS-DOSに渡す。
345:
346: void d_entry( )
347: {
348:     if (packet->cmd > 9)
349:         packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_BADCMD); /* 1/O リクエスト・コマンド・
350:         コードが間違っている */
351:
352:     else {
353:         (*func[packet->cmd])( ); /* 処理に分岐する */
354:     }
355:     return;
356: }
357:
358:
359:
360:
361: /* イニシャライズ処理
362:
363: /* PD7211のイニシャライズ及びHDDのイニシャライズを 入力: prog_off, prog_seg 出力: packet
364: /* 行う。
365:
366: void init( )
367: {
368:     int status;
369:     char c;
370:
371:     io_init( );
372:     if (seg1!=0)
373:         /* エラー発生か? */
374:
375:         packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL); /* その他のエラー */
376:         return;
377:
378:     if (ready_h != 0)
379:         /* レディ状態か? */
380:         return;
381:
382:     bios_msc( );
383:     if ((status = stjudg( )) != 0)
384:         /* BIOSモード・センス・コマンド処理*/
385:         /* コマンド正常終了か? */
386:         packet->command_status = status;
387:         return;
388:
389:     c = 0;
390:
391:     /* モード・センス・データと希望のモード・センス・データの比較処理 */
392:     while (c < 64)
393:         if (dha_msd[c] == dha_msd_c(c))

```

```

394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:

```

394: {  
 395: c++;  
 396: }  
 397: else  
 398: {  
 399: bios\_msc( );  
 400: if ((status = stjudg( )) == 0)  
 401: /\* BIOSモード・センス・コマンド処理\*/  
 402: /\* コマンド正常終了か? \*/  
 403: break;  
 404: }  
 405: else  
 406: {  
 407: packet->command\_status = status;  
 408: return;  
 409: }  
 410: }  
 411: packet->up.in.unit = 1;  
 412: packet->up.in.end\_off = prog\_off;  
 413: packet->up.in.end\_seg = prog\_seg;  
 414: packet->command\_status = ST\_DONE;  
 415: return;  
 416: }  
 417:
 418: /\* BPBの配列へのポインタをセット \*/  
 419: packet->up.in.bpb.dj.off = bbbpoin;  
 420: packet->up.in.bpb.dj.seg = prog\_seg;  
 421: packet->command\_status = ST\_DONE;  
 422: return;  
 423: }  
 424:
 425: /\* MS-DOSに "メディアが変更されていない" を返す。  
 426: /\* 入力: なし  
 427: /\* 出力: packet  
 428:
 429: void media\_check( )  
 430: {  
 431: packet->up.mc.retrpr = 1;  
 432: packet->command\_status = ST\_DONE;  
 433: return;  
 434: }  
 435:
 436:
 437:
 438:
 439:
 440:
 441:
 442:
 443:
 444:
 445: void build\_bpb( )  
 446: {  
 447: if (bpb\_input() == 0) {  
 448: }  
 449: packet->up.bb.mediaab = bpb\_h.med.id; /\* DPBへのメディア・ディスクリプタ・バイトをセット \*/  
 450: packet->up.bb.bpb.off = bbbp.in;  
 451: packet->up.bb.bpb.seg = prog\_seg;  
 452: packet->command\_status = ST\_DONE;  
 453: return;  
 454: }  
 455: }  
 456:
 457:
 458:
 459:
 460:
 461:
 462:
 463:
 464:
 465: void error\_io( )  
 466: {  
 467: packet->command\_status = (ST\_ERROR|ST\_DONE|ST\_ERR\_BADCMD); /\* 1/O リクエスト・  
 468: コマンド・コードが間違っている \*/  
 469: return;  
 470: }



```

471: /* **** インフラット処理 **** */
472: /* リード・エクステンディッド・コマンドを送信して、 */
473: /* ハード・ディスク・ドライブからデータを */
474: /* リードする。 */
475: /* **** */
476: void input()
477: {
478:     int status;
479:     bios_packet_set();
480:     if((status = scsi_read()) == 0)
481:     {
482:         packet->command_status = ST_DONE;
483:         /* リード・コマンド設定処理 */
484:         /* BIOSリード・コマンド処理 */
485:         /* コマンド正常終了か? */
486:         /* エラー */
487:         return;
488:     }
489:     packet->command_status = status;
490:     return;
491: }
492: /* **** アウトフラット処理 **** */
493: /* ライト・エクステンディッド・コマンドを送信して、 */
494: /* ハード・ディスク・ドライブにデータをライツする。 */
495: /* **** */
496: void output()
497: {
498:     int status;
499:     bios_packet_set();
500:     if((status = scsi_write()) == 0)
501:     {
502:         packet->command_status = ST_DONE;
503:         /* ライト・コマンド設定処理 */
504:         /* BIOSライツ・コマンド処理 */
505:         /* コマンド正常終了か? */
506:         /* エラー */
507:         return;
508:     }
509:     packet->command_status = status;
510:     return;
511: }
512: /* **** アウトフラット・ウィズ・ベリファイ処理 **** */
513: /* ライト・アンド・ベリファイ・コマンドを送信して、ハード・ディスク */
514: /* ライト・ドライブにデータをライツした後にデータをベリファイする。 */
515: /* **** */
516: void output_v()
517: {
518:     int status;
519:     bios_packet_set();
520:     if((status = scsi_writelv()) == 0)
521:     {
522:         packet->command_status = ST_DONE;
523:         /* ライト・アンド・ベリファイ・コマンド設定処理 */
524:         /* BIOSライツ・アンド・ベリファイ処理 */
525:         /* コマンド正常終了か? */
526:         /* エラー */
527:         return;
528:     }
529:     packet->command_status = status;
530:     return;
531: }
532: /* **** ステータス判断処理 **** */
533: /* PMD 2.1.1 制御処理の結果 (scsi1, scsi12, */
534: /* status) によってエラー時は1、正常時は0を上位桁に戻す。 */
535: /* **** */
536: int stjjud()

```

```

537: {
538:     int status;
539:     if(scsi1==0)
540:     {
541:         return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);
542:     }
543:     if(scsi12==0)
544:     {
545:         bios_rsc();
546:         if(scsi1==0)
547:         {
548:             return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);
549:         }
550:         if(scsi12==0)
551:         {
552:             return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);
553:         }
554:         return(ST_ERROR|ST_DONE|ST_ERR_GENERAL);
555:     }
556:     /* センズ・コードに於ける処理 */
557:     switch(dha_rsd[12])
558:     {
559:         case R_R:
560:             /* リード・コマンド・エラー */
561:             /* リード・コマンド・エラー */
562:             /* リード・コマンド・エラー */
563:             /* リード・コマンド・エラー */
564:             /* リード・コマンド・エラー */
565:             /* リード・コマンド・エラー */
566:             /* リード・コマンド・エラー */
567:             /* リード・コマンド・エラー */
568:             /* リード・コマンド・エラー */
569:             /* リード・コマンド・エラー */
570:             /* リード・コマンド・エラー */
571:             /* リード・コマンド・エラー */
572:             /* リード・コマンド・エラー */
573:             /* リード・コマンド・エラー */
574:             /* リード・コマンド・エラー */
575:             /* リード・コマンド・エラー */
576:             /* リード・コマンド・エラー */
577:             /* リード・コマンド・エラー */
578:             /* リード・コマンド・エラー */
579:             /* リード・コマンド・エラー */
580:             /* リード・コマンド・エラー */
581:             /* リード・コマンド・エラー */
582:             /* リード・コマンド・エラー */
583:             /* リード・コマンド・エラー */
584:             /* リード・コマンド・エラー */
585:             /* リード・コマンド・エラー */
586:             /* リード・コマンド・エラー */
587:             /* リード・コマンド・エラー */
588:             /* リード・コマンド・エラー */
589:             /* リード・コマンド・エラー */
590:             /* リード・コマンド・エラー */
591:             /* リード・コマンド・エラー */
592:             /* リード・コマンド・エラー */
593:             /* リード・コマンド・エラー */
594:             /* リード・コマンド・エラー */
595:             /* リード・コマンド・エラー */
596:             /* リード・コマンド・エラー */
597:             /* リード・コマンド・エラー */
598:             /* リード・コマンド・エラー */
599:             /* リード・コマンド・エラー */
600:             /* リード・コマンド・エラー */
601:             /* リード・コマンド・エラー */
602:             /* リード・コマンド・エラー */
603:             /* リード・コマンド・エラー */
604:             /* リード・コマンド・エラー */
605:             /* リード・コマンド・エラー */
606:             /* リード・コマンド・エラー */
607:             /* リード・コマンド・エラー */
608:             /* リード・コマンド・エラー */
609:             /* リード・コマンド・エラー */
610:             /* リード・コマンド・エラー */
611:             /* リード・コマンド・エラー */
612:             /* リード・コマンド・エラー */
613:             /* リード・コマンド・エラー */
614:             /* リード・コマンド・エラー */
615:             /* リード・コマンド・エラー */
616:             /* リード・コマンド・エラー */
617:             /* リード・コマンド・エラー */
618:             /* リード・コマンド・エラー */
619:             /* リード・コマンド・エラー */
620:             /* リード・コマンド・エラー */
621:             /* リード・コマンド・エラー */
622:             /* リード・コマンド・エラー */
623:             /* リード・コマンド・エラー */
624:             /* リード・コマンド・エラー */
625:             /* リード・コマンド・エラー */
626:             /* リード・コマンド・エラー */

```

```

571: { packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL); /* その他のエラー */
572: return(1);
573: }
574: if(sttel=0)
575: {
576: bios_res( ); /* グッド・コンディションか? */
577: if(sscif=0) /* モード・センス・コマンド処理 */
578: /* エラー発生か? */
579: {
580: packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL); /* その他のエラー */
581: return(1);
582: }
583: if(sttel=0)
584: {
585: packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL); /* その他のエラー */
586: return(1);
587: }
588: /* セン・ス・コードに対する処理 */
589: switch(dha_rsd[12])
590: {
591: case N_TOP: /* N-127 0 フォインド */
592: case P_DP: /* N-127 8 f77 35477147- */
593: packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_NOTROY);
594: return(1);
595: case P_O: /* N-127 9 4477 N-127 N-127 127 */
596: bios_msc( ); /* モード・セレクト・コマンド処理 */
597: if(sscif=0) /* エラー発生か? */
598: {
599: packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);
600: return(1);
601: }
602: if(ready_hk(==0) /* レディ状態か? (再帰) */
603: {
604: return(0);
605: }
606: else
607: {
608: return(1);
609: }
610: case N_R: /* N-127 147-エニフ・コマンド処理 */
611: bios_ssuc( ); /* エラー発生か? */
612: if(sscif=0)
613: {
614: packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL);
615: return(1);
616: }
617: if(ready_hk(==0) /* レディ状態か? (再帰) */
618: {
619: return(0);
620: }
621: else
622: {
623: return(1);
624: }
625: default:
626: {
627: packet->command_status = (ST_ERROR|ST_DONE|ST_ERR_GENERAL); /* その他のエラー */
628: return(1);
629: }
630: }
631: return(0);
632: }
633: }
634: }
635: }
636: }
637: }
638: }
639: }
640: }
641: }
642: }
643: }
644: }
645: }
646: }
647: }
648: }
649: }
650: }
651: }
652: }
653: }
654: }
655: }
656: }
657: }
658: }
659: }
660: }
661: }
662: }
663: }
664: }
665: }
666: }
667: }
668: }
669: }
670: }
671: }
672: }
673: }
674: }
675: }
676: }
677: }
678: }
679: }
680: }
681: }
682: }
683: }
684: }
685: }
686: }
687: }
688: }
689: }
690: }
691: }
692: }
693: }
694: }
695: }
696: }
697: }
698: }
699: }
700: }
701: }
702: }
703: }
704: }
705: }
706: }
707: }
708: }
709: }
710: }
711: }
712: }
713: }
714: }
715: }
716: }
717: }
718: }
719: }
720: }
721: }
722: }
723: }
724: }
725: }
726: }
727: }
728: }
729: }
730: }
731: }
732: }
733: }
734: }
735: }
736: }
737: }
738: }
739: }
740: }
741: }
742: }
743: }
744: }
745: }
746: }
747: }
748: }
749: }
750: }
751: }
752: }
753: }
754: }
755: }
756: }
757: }
758: }
759: }
760: }
761: }
762: }
763: }
764: }
765: }
766: }
767: }
768: }
769: }
770: }
771: }
772: }
773: }
774: }
775: }
776: }
777: }
778: }
779: }
780: }
781: }
782: }
783: }
784: }
785: }
786: }
787: }
788: }
789: }
790: }
791: }
792: }
793: }
794: }
795: }
796: }
797: }
798: }
799: }
800: }
801: }
802: }
803: }
804: }
805: }
806: }
807: }
808: }
809: }
810: }
811: }
812: }
813: }
814: }
815: }
816: }
817: }
818: }
819: }
820: }
821: }
822: }
823: }
824: }
825: }
826: }
827: }
828: }
829: }
830: }
831: }
832: }
833: }
834: }
835: }
836: }
837: }
838: }
839: }
840: }
841: }
842: }
843: }
844: }
845: }
846: }
847: }
848: }
849: }
850: }
851: }
852: }
853: }
854: }
855: }
856: }
857: }
858: }
859: }
860: }
861: }
862: }
863: }
864: }
865: }
866: }
867: }
868: }
869: }
870: }
871: }
872: }
873: }
874: }
875: }
876: }
877: }
878: }
879: }
880: }
881: }
882: }
883: }
884: }
885: }
886: }
887: }
888: }
889: }
890: }
891: }
892: }
893: }
894: }
895: }
896: }
897: }
898: }
899: }
900: }
901: }
902: }
903: }
904: }
905: }
906: }
907: }
908: }
909: }
910: }
911: }
912: }
913: }
914: }
915: }
916: }
917: }
918: }
919: }
920: }
921: }
922: }
923: }
924: }
925: }
926: }
927: }
928: }
929: }
930: }
931: }
932: }
933: }
934: }
935: }
936: }
937: }
938: }
939: }
940: }
941: }
942: }
943: }
944: }
945: }
946: }
947: }
948: }
949: }
950: }
951: }
952: }
953: }
954: }
955: }
956: }
957: }
958: }
959: }
960: }
961: }
962: }
963: }
964: }
965: }
966: }
967: }
968: }
969: }
970: }
971: }
972: }
973: }
974: }
975: }
976: }
977: }
978: }
979: }
980: }
981: }
982: }
983: }
984: }
985: }
986: }
987: }
988: }
989: }
990: }
991: }
992: }
993: }
994: }
995: }
996: }
997: }
998: }
999: }
1000: }
1001: }
1002: }
1003: }
1004: }
1005: }
1006: }
1007: }
1008: }
1009: }
1010: }
1011: }
1012: }
1013: }
1014: }
1015: }
1016: }
1017: }
1018: }
1019: }
1020: }
1021: }
1022: }
1023: }
1024: }
1025: }
1026: }
1027: }
1028: }
1029: }
1030: }
1031: }
1032: }
1033: }
1034: }
1035: }
1036: }
1037: }
1038: }
1039: }
1040: }
1041: }
1042: }
1043: }
1044: }
1045: }
1046: }
1047: }
1048: }
1049: }
1050: }
1051: }
1052: }
1053: }
1054: }
1055: }
1056: }
1057: }
1058: }
1059: }
1060: }
1061: }
1062: }
1063: }
1064: }
1065: }
1066: }
1067: }
1068: }
1069: }
1070: }
1071: }
1072: }
1073: }
1074: }
1075: }
1076: }
1077: }
1078: }
1079: }
1080: }
1081: }
1082: }
1083: }
1084: }
1085: }
1086: }
1087: }
1088: }
1089: }
1090: }
1091: }
1092: }
1093: }
1094: }
1095: }
1096: }
1097: }
1098: }
1099: }
1100: }
1101: }
1102: }
1103: }
1104: }
1105: }
1106: }
1107: }
1108: }
1109: }
1110: }
1111: }
1112: }
1113: }
1114: }
1115: }
1116: }
1117: }
1118: }
1119: }
1120: }
1121: }
1122: }
1123: }
1124: }
1125: }
1126: }
1127: }
1128: }
1129: }
1130: }
1131: }
1132: }
1133: }
1134: }
1135: }
1136: }
1137: }
1138: }
1139: }
1140: }
1141: }
1142: }
1143: }
1144: }
1145: }
1146: }
1147: }
1148: }
1149: }
1150: }
1151: }
1152: }
1153: }
1154: }
1155: }
1156: }
1157: }
1158
```

```

706: *(unsigned int *)&(bios_packet.sector_count_1)) += bsctn;
707: bios_packet.start_sector_1 = packet->up.rw.start_sector_1; /* 開始セクタ番号セット */
708: bios_packet.start_sector_hl = packet->up.rw.start_sector_m;
709: bios_packet.start_sector_m = bios_packet.start_sector_lm = 0;
710: *(unsigned long *)&(bios_packet.start_sector_1)) += bsctn;
711: return;
712: }
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965:
966:
967:
968:
969:
970:
971:
972:
973:
974:
975:
976:
977:
978:
979:
980:
981:
982:
983:
984:
985:
986:
987:
988:
989:
990:
991:
992:
993:
994:
995:
996:
997:
998:
999:
1000:
1001:
1002:
1003:
1004:
1005:
1006:
1007:
1008:
1009:
1010:
1011:
1012:
1013:
1014:
1015:
1016:
1017:
1018:
1019:
1020:
1021:
1022:
1023:
1024:
1025:
1026:
1027:
1028:
1029:
1030:
1031:
1032:
1033:
1034:
1035:
1036:
1037:
1038:
1039:
1040:
1041:
1042:
1043:
1044:
1045:
1046:
1047:
1048:
1049:
1050:
1051:
1052:
1053:
1054:
1055:
1056:
1057:
1058:
1059:
1060:
1061:
1062:
1063:
1064:
1065:
1066:
1067:
1068:
1069:
1070:
1071:
1072:
1073:
1074:
1075:
1076:
1077:
1078:
1079:
1080:
1081:
1082:
1083:
1084:
1085:
1086:
1087:
1088:
1089:
1090:
1091:
1092:
1093:
1094:
1095:
1096:
1097:
1098:
1099:
1100:
1101:
1102:
1103:
1104:
1105:
1106:
1107:
1108:
1109:
1110:
1111:
1112:
1113:
1114:
1115:
1116:
1117:
1118:
1119:
1120:
1121:
1122:
1123:
1124:
1125:
1126:
1127:
1128:
1129:
1130:
1131:
1132:
1133:
1134:
1135:
1136:
1137:
1138:
1139:
1140:
1141:
1142:
1143:
1144:
1145:
1146:
1147:
1148:
1149:
1150:
1151:
1152:
1153:
1154:
1155:
1156:
1157:
1158:
1159:
1160:
1161:
1162:
1163:
1164:
1165:
1166:
1167:
1168:
1169:
1170:
1171:
1172:
1173:
1174:
1175:
1176:
1177:
1178:
1179:
1180:
1181:
1182:
1183:
1184:
1185:
1186:
1187:
1188:
1189:
1190:
1191:
1192:
1193:
1194:
1195:
1196:
1197:
1198:
1199:
1200:
1201:
1202:
1203:
1204:
1205:
1206:
1207:
1208:
1209:
1210:
1211:
1212:
1213:
1214:
1215:
1216:
1217:
1218:
1219:
1220:
1221:
1222:
1223:
1224:
1225:
1226:
1227:
1228:
1229:
1230:
1231:
1232:
1233:
1234:
1235:
1236:
1237:
1238:
1239:
1240:
1241:
1242:
1243:
1244:
1245:
1246:
1247:
1248:
1249:
1250:
1251:
1252:
1253:
1254:
1255:
1256:
1257:
1258:
1259:
1260:
1261:
1262:
1263:
1264:
1265:
1266:
1267:
1268:
1269:
1270:
1271:
1272:
1273:
1274:
1275:
1276:
1277:
1278:
1279:
1280:
1281:
1282:
1283:
1284:
1285:
1286:
1287:
1288:
1289:
1290:
1291:
1292:
1293:
1294:
1295:
1296:
1297:
1298:
1299:
1300:
1301:
1302:
1303:
1304:
1305:
1306:
1307:
1308:
1309:
1310:
1311:
1312:
1313:
1314:
1315:
1316:
1317:
1318:
1319:
1320:
1321:
1322:
1323:
1324:
1325:
1326:
1327:
1328:
1329:
1330:
1331:
1332:
1333:
1334:
1335:
1336:
1337:
1338:
1339:
1340:
1341:
1342:
1343:
1344:
1345:
1346:
1347:
1348:
1349:
1350:
1351:
1352:
1353:
1354:
1355:
1356:
1357:
1358:
1359:
1360:
1361:
1362:
1363:
1364:
1365:
1366:
1367:
1368:
1369:
1370:
1371:
1372:
1373:
1374:
1375:
1376:
1377:
1378:
1379:
1380:
1381:
1382:
1383:
1384:
1385:
1386:
1387:
1388:
1389:
1390:
1391:
1392:
1393:
1394:
1395:
1396:
1397:
1398:
1399:
1400:
1401:
1402:
1403:
1404:
1405:
1406:
1407:
1408:
1409:
1410:
1411:
1412:
1413:
1414:
1415:
1416:
1417:
1418:
1419:
1420:
```



```

783: tcc = SSVCI_BC; /* トランスファ・コマンド・カウンタをセット */
784: return;
785: }
786:
787: /*****
788: * テスト・ユニット・レディ・コマンド設定処理
789: *
790: * テスト・ユニット・レディ・コマンドのポイントを cha に、
791: * バイト数を tcc に設定する。
792: *
793: * 入力: なし
794: * 出力: cha, tcc
795: */
796: void turec (
797:     /* 入力 */
798:     char cha,
799:     /* 出力 */
800:     int tcc)
801: {
802:     cha.cha = &cha_turec[0];
803:     tcc = TUREC_BC;
804:     return;
805: }
806:
807: /*****
808: * リード・エクステンディッド・コマンド設定処理
809: *
810: * リード・エクステンディッド・コマンドのポイントを cha に、バイト
811: * 数を tcc に設定する。
812: *
813: * リード・エクステンディッド・データのポイントを dha_seg,
814: * おおびdha_off に、バイト数を ldc に設定する。
815: */
816: void tlec (
817:     /* 入力 */
818:     char cha,
819:     /* 出力 */
820:     int tcc,
821:     /* 入力 */
822:     char dha_seg,
823:     /* 出力 */
824:     int dha_off,
825:     /* 入力 */
826:     int ldc)
827: {
828:     cha.cha = &cha_tlec;
829:     tcc = REC_BC;
830:     dha_off = bios_packet.transw_off;
831:     dha_seg = bios_packet.transw_seg;
832:     cha.rec.trans_sector_m = bios_packet.sector_count;
833:     cha.rec.trans_sector_l = bios_packet.sector_count_l;
834:     ldc = *((int *)(&bios_packet.sector_count_l)) * 512;
835:     cha.rec.start_sector_m = bios_packet.start_sector_hi;
836:     cha.rec.start_sector_l = bios_packet.start_sector_lo;
837:     cha.rec.start_sen_hi = bios_packet.start_sector_hi;
838:     cha.rec.start_sen_lo = bios_packet.start_sector_lo;
839:     return;
840: }
841:
842: /*****
843: * ライト・エクステンディッド・コマンド設定処理
844: *
845: * ライト・エクステンディッド・コマンドのポイントを cha に、バイト
846: * 数を tcc に設定する。
847: *
848: * ライト・エクステンディッド・データのポイントを dha_seg,
849: * おおびdha_off に、バイト数を ldc に設定する。
850: */
851: void tlec (
852:     /* 入力 */
853:     char cha,
854:     /* 出力 */
855:     int tcc,
856:     /* 入力 */
857:     char dha_seg,
858:     /* 出力 */
859:     int dha_off,
860:     /* 入力 */
861:     int ldc)
862: {
863:     cha.cha = &cha_tlec;
864:     tcc = REC_BC;
865:     dha_off = bios_packet.transw_off;
866:     dha_seg = bios_packet.transw_seg;
867:     cha.wec.trans_sector_m = bios_packet.sector_count;
868:     cha.wec.trans_sector_l = bios_packet.sector_count_l;
869:     ldc = *((int *)(&bios_packet.sector_count_l)) * 512;
870:     cha.wec.start_sector_m = bios_packet.start_sector_hi;
871:     cha.wec.start_sector_l = bios_packet.start_sector_lo;
872:     cha.wec.start_sen_hi = bios_packet.start_sector_hi;
873:     cha.wec.start_sen_lo = bios_packet.start_sector_lo;
874:     return;
875: }
876:
877: /*****
878: * コマンド先頭アドレスをセット
879: *
880: * トランスファ・コマンド・カウンタをセット
881: */
882: void tccs (
883:     /* 入力 */
884:     int tcc)
885: {
886:     tcc = SSVCI_BC;
887:     return;
888: }
889:
890: /*****
891: * コマンド先頭アドレスをセット
892: *
893: * トランスファ・コマンド・カウンタをセット
894: */
895: void tccs (
896:     /* 入力 */
897:     int tcc)
898: {
899:     tcc = SSVCI_BC;
900:     return;
901: }
902:
903: /*****
904: * コマンド先頭アドレスをセット
905: *
906: * トランスファ・コマンド・カウンタをセット
907: */
908: void tccs (
909:     /* 入力 */
910:     int tcc)
911: {
912:     tcc = SSVCI_BC;
913:     return;
914: }
915:
916: /*****
917: * コマンド先頭アドレスをセット
918: *
919: * トランスファ・コマンド・カウンタをセット
920: */
921: void tccs (
922:     /* 入力 */
923:     int tcc)
924: {
925:     tcc = SSVCI_BC;
926:     return;
927: }
928:
929: /*****
930: * コマンド先頭アドレスをセット
931: *
932: * トランスファ・コマンド・カウンタをセット
933: */
934: void tccs (
935:     /* 入力 */
936:     int tcc)
937: {
938:     tcc = SSVCI_BC;
939:     return;
940: }
941:
942: /*****
943: * コマンド先頭アドレスをセット
944: *
945: * トランスファ・コマンド・カウンタをセット
946: */
947: void tccs (
948:     /* 入力 */
949:     int tcc)
950: {
951:     tcc = SSVCI_BC;
952:     return;
953: }
954:
955: /*****
956: * コマンド先頭アドレスをセット
957: *
958: * トランスファ・コマンド・カウンタをセット
959: */
960: void tccs (
961:     /* 入力 */
962:     int tcc)
963: {
964:     tcc = SSVCI_BC;
965:     return;
966: }
967:
968: /*****
969: * コマンド先頭アドレスをセット
970: *
971: * トランスファ・コマンド・カウンタをセット
972: */
973: void tccs (
974:     /* 入力 */
975:     int tcc)
976: {
977:     tcc = SSVCI_BC;
978:     return;
979: }
980:
981: /*****
982: * コマンド先頭アドレスをセット
983: *
984: * トランスファ・コマンド・カウンタをセット
985: */
986: void tccs (
987:     /* 入力 */
988:     int tcc)
989: {
990:     tcc = SSVCI_BC;
991:     return;
992: }
993:
994: /*****
995: * コマンド先頭アドレスをセット
996: *
997: * トランスファ・コマンド・カウンタをセット
998: */
999: void tccs (
1000:     /* 入力 */
1001:     int tcc)
1002: {
1003:     tcc = SSVCI_BC;
1004:     return;
1005: }
1006:
1007: /*****
1008: * コマンド先頭アドレスをセット
1009: *
1010: * トランスファ・コマンド・カウンタをセット
1011: */
1012: void tccs (
1013:     /* 入力 */
1014:     int tcc)
1015: {
1016:     tcc = SSVCI_BC;
1017:     return;
1018: }
1019:
1020: /*****
1021: * コマンド先頭アドレスをセット
1022: *
1023: * トランスファ・コマンド・カウンタをセット
1024: */
1025: void tccs (
1026:     /* 入力 */
1027:     int tcc)
1028: {
1029:     tcc = SSVCI_BC;
1030:     return;
1031: }
1032:
1033: /*****
1034: * コマンド先頭アドレスをセット
1035: *
1036: * トランスファ・コマンド・カウンタをセット
1037: */
1038: void tccs (
1039:     /* 入力 */
1040:     int tcc)
1041: {
1042:     tcc = SSVCI_BC;
1043:     return;
1044: }
1045:
1046: /*****
1047: * コマンド先頭アドレスをセット
1048: *
1049: * トランスファ・コマンド・カウンタをセット
1050: */
1051: void tccs (
1052:     /* 入力 */
1053:     int tcc)
1054: {
1055:     tcc = SSVCI_BC;
1056:     return;
1057: }
1058:
1059: /*****
1060: * コマンド先頭アドレスをセット
1061: *
1062: * トランスファ・コマンド・カウンタをセット
1063: */
1064: void tccs (
1065:     /* 入力 */
1066:     int tcc)
1067: {
1068:     tcc = SSVCI_BC;
1069:     return;
1070: }
1071:
1072: /*****
1073: * コマンド先頭アドレスをセット
1074: *
1075: * トランスファ・コマンド・カウンタをセット
1076: */
1077: void tccs (
1078:     /* 入力 */
1079:     int tcc)
1080: {
1081:     tcc = SSVCI_BC;
1082:     return;
1083: }
1084:
1085: /*****
1086: * コマンド先頭アドレスをセット
1087: *
1088: * トランスファ・コマンド・カウンタをセット
1089: */
1090: void tccs (
1091:     /* 入力 */
1092:     int tcc)
1093: {
1094:     tcc = SSVCI_BC;
1095:     return;
1096: }
1097:
1098: /*****
1099: * コマンド先頭アドレスをセット
1100: *
1101: * トランスファ・コマンド・カウンタをセット
1102: */
1103: void tccs (
1104:     /* 入力 */
1105:     int tcc)
1106: {
1107:     tcc = SSVCI_BC;
1108:     return;
1109: }
1110:
1111: /*****
1112: * コマンド先頭アドレスをセット
1113: *
1114: * トランスファ・コマンド・カウンタをセット
1115: */
1116: void tccs (
1117:     /* 入力 */
1118:     int tcc)
1119: {
1120:     tcc = SSVCI_BC;
1121:     return;
1122: }
1123:
1124: /*****
1125: * コマンド先頭アドレスをセット
1126: *
1127: * トランスファ・コマンド・カウンタをセット
1128: */
1129: void tccs (
1130:     /* 入力 */
1131:     int tcc)
1132: {
1133:     tcc = SSVCI_BC;
1134:     return;
1135: }
1136:
1137: /*****
1138: * コマンド先頭アドレスをセット
1139: *
1140: * トランスファ・コマンド・カウンタをセット
1141: */
1142: void tccs (
1143:     /* 入力 */
1144:     int tcc)
1145: {
1146:     tcc = SSVCI_BC;
1147:     return;
1148: }
1149:
1150: /*****
1151: * コマンド先頭アドレスをセット
1152: *
1153: * トランスファ・コマンド・カウンタをセット
1154: */
1155: void tccs (
1156:     /* 入力 */
1157:     int tcc)
1158: {
1159:     tcc = SSVCI_BC;
1160:     return;
1161: }
1162:
1163: /*****
1164: * コマンド先頭アドレスをセット
1165: *
1166: * トランスファ・コマンド・カウンタをセット
1167: */
1168: void tccs (
1169:     /* 入力 */
1170:     int tcc)
1171: {
1172:     tcc = SSVCI_BC;
1173:     return;
1174: }
1175:
1176: /*****
1177:
```

```

860: 860: 入力: bios_packet
861: 861: 出力: cha, tcc, tdc,
862: 862: dha_seg, dha_off
863: 863:
864: 864: ライト・アンド・ペリフアイ・データのポイントをdha_seg.
865: 865: ライト・アンド・ペリフアイ・データのポイントをdha_seg.
866: 866: おおびdha_offに, バイト数をtdcに設定する.
867: 867:
868: 868: void char( )
869: 869: {
870: 870:     cha, cha_st = &cha_wavc;
871: 871:     tcc = &tcc_buf;
872: 872:     dha_off = bios_packet.transw_off;
873: 873:     dha_seg = bios_packet.transw_seg;
874: 874:     cha_wavc.trans_sector_m = bios_packet.sector_count_m;
875: 875:     cha_wavc.trans_sector_l = bios_packet.sector_count_l;
876: 876:
877: 877:     tdc = *(int *)&(bios_packet.sector_count_l) * 512; /* トランスファ・データ・カウンタをセット */
878: 878:     cha_wavc.start_scn_hl = bios_packet.start_sector_hl; /* 開始セクタ番号セット */
879: 879:     cha_wavc.start_scn_l = bios_packet.start_sector_l;
880: 880:     cha_wavc.start_scn_hm = bios_packet.start_sector_hm;
881: 881:     cha_wavc.start_scn_lm = bios_packet.start_sector_lm;
882: 882:
883: 883:     return;
884: 884: }
885: 885:
886: 886:
887: 887:
888: 888:
889: 889:
890: 890: int bpb_input( )
891: 891: {
892: 892:     bios_bpb_rec( );
893: 893:     scsi_read( );
894: 894:     return(stjwd( ));
895: 895: }
896: 896:
897: 897:
898: 898:
899: 899:
900: 900: void bios_bpb_rec( )
901: 901: {
902: 902:     bios_packet.transw_off = &work_buf; /* 転送アドレスをセット */
903: 903:     bios_packet.transw_seg = prog_seg;
904: 904:
905: 905:     bios_packet.sector_count_m = 0; /* 転送セクタ数セット */
906: 906:     bios_packet.sector_count_l = 1;
907: 907:
908: 908:     bios_packet.start_sector_hl = bios_packet.start_sector_l = 0; /* 開始セクタ番号セット(先頭=0) */
909: 909:     bios_packet.start_sector_hm = bios_packet.start_sector_lm = 0;
910: 910:
911: 911:     return;
912: 912: }
913: 913:
914: 914:
915: 915:
916: 916: void bpb_set( )
917: 917: {
918: 918:     *sptr;
919: 919:     char *dptr;
920: 920:     int i;
921: 921:
922: 922:     sptr = (char *)&work_buf;
923: 923:     dptr = (char *)&bpb_h;
924: 924:     for(i = 0; i < 13; ++i) {
925: 925:         *bpb_ptr++ = *(sptr+++i+1);
926: 926:     }
927: 927: }
928: 928:
929: 929:
930: 930:
931: 931:
932: 932:
933: 933:
934: 934:
935: 935: int bios_resc( )
936: 936: {
937: 937:
938: 938:
939: 939:
940: 940:
941: 941:
942: 942:
943: 943:
944: 944:
945: 945:
946: 946:
947: 947:
948: 948:
949: 949:
950: 950:
951: 951:
952: 952:
953: 953:
954: 954:
955: 955:
956: 956:
957: 957:
958: 958:
959: 959:
960: 960:
961: 961:
962: 962:
963: 963:
964: 964:
965: 965:
966: 966:
967: 967:
968: 968:
969: 969:
970: 970:
971: 971:
972: 972:
973: 973:
974: 974:
975: 975:
976: 976:
977: 977:
978: 978:
979: 979:
980: 980:
981: 981:
982: 982:
983: 983:
984: 984:
985: 985:
986: 986:
987: 987:
988: 988:
989: 989:
990: 990:
991: 991:
992: 992:
993: 993:
994: 994:
995: 995:
996: 996:
997: 997:
998: 998:
999: 999:
1000: 1000:
1001: 1001:
1002: 1002:
1003: 1003:
1004: 1004:
1005: 1005:
1006: 1006:
1007: 1007:
1008: 1008:
1009: 1009:
1010: 1010:
1011: 1011:
1012: 1012:
1013: 1013:
1014: 1014:
1015: 1015:
1016: 1016:
1017: 1017:
1018: 1018:
1019: 1019:
1020: 1020:
1021: 1021:
1022: 1022:
1023: 1023:
1024: 1024:
1025: 1025:
1026: 1026:
1027: 1027:
1028: 1028:
1029: 1029:
1030: 1030:
1031: 1031:
1032: 1032:
1033: 1033:
1034: 1034:
1035: 1035:
1036: 1036:
1037: 1037:
1038: 1038:
1039: 1039:
1040: 1040:
1041: 1041:
1042: 1042:
1043: 1043:
1044: 1044:
1045: 1045:
1046: 1046:
1047: 1047:
1048: 1048:
1049: 1049:
1050: 1050:
1051: 1051:
1052: 1052:
1053: 1053:
1054: 1054:
1055: 1055:
1056: 1056:
1057: 1057:
1058: 1058:
1059: 1059:
1060: 1060:
1061: 1061:
1062: 1062:
1063: 1063:
1064: 1064:
1065: 1065:
1066: 1066:
1067: 1067:
1068: 1068:
1069: 1069:
1070: 1070:
1071: 1071:
1072: 1072:
1073: 1073:
1074: 1074:
1075: 1075:
1076: 1076:
1077: 1077:
1078: 1078:
1079: 1079:
1080: 1080:
1081: 1081:
1082: 1082:
1083: 1083:
1084: 1084:
1085: 1085:
1086: 1086:
1087: 1087:
1088: 1088:
1089: 1089:
1090: 1090:
1091: 1091:
1092: 1092:
1093: 1093:
1094: 1094:
1095: 1095:
1096: 1096:
1097: 1097:
1098: 1098:
1099: 1099:
1100: 1100:
1101: 1101:
1102: 1102:
1103: 1103:
1104: 1104:
1105: 1105:
1106: 1106:
1107: 1107:
1108: 1108:
1109: 1109:
1110: 1110:
1111: 1111:
1112: 1112:
1113: 1113:
1114: 1114:
1115: 1115:
1116: 1116:
1117: 1117:
1118: 1118:
1119: 1119:
1120: 1120:
1121: 1121:
1122: 1122:
1123: 1123:
1124: 1124:
1125: 1125:
1126: 1126:
1127: 1127:
1128: 1128:
1129: 1129:
1130: 1130:
1131: 1131:
1132: 1132:
1133: 1133:
1134: 1134:
1135: 1135:
1136: 1136:
1137: 1137:
1138: 1138:
1139: 1139:
1140: 1140:
1141: 1141:
1142: 1142:
1143: 1143:
1144: 1144:
1145: 1145:
1146: 1146:
1147: 1147:
1148: 1148:
1149: 1149:
1150: 1150:
1151: 1151:
1152: 1152:
1153: 1153:
1154: 1154:
1155: 1155:
1156: 1156:
1157: 1157:
1158: 1158:
1159: 1159:
1160: 1160:
1161: 1161:
1162: 1162:
1163: 1163:
1164: 1164:
1165: 1165:
1166: 1166:
1167: 1167:
1168: 1168:
1169: 1169:
1170: 1170:
1171: 1171:
1172: 1172:
1173: 1173:
1174: 1174:
1175: 1175:
1176: 1176:
1177: 1177:
1178: 1178:
1179: 1179:
1180: 1180:
1181: 1181:
```

```

937:         rsc0(); /* リクエスト・センス・コマンド設定 */
938:         scsi_bios(); /* μPD72111制御処理 */
939:         return((int)sste);
940:     }
941: }
942:
943: /*
944:  * BIOSモード・ファンデーション処理
945:  */
946: int bios_msc()
947: {
948:     msc0(); /* モード・センス・コマンド設定処理 */
949:     scsi_bios(); /* μPD72111制御処理 */
950:     return((int)sste);
951: }
952:
953: /*
954:  * BIOSモード・セレクト・コマンド処理
955:  */
956: int bios_msc1()
957: {
958:     msc1(); /* モード・セレクト・コマンド設定処理 */
959:     scsi_bios(); /* μPD72111制御処理 */
960:     return((int)sste);
961: }
962:
963: /*
964:  * BIOSスタート/ストップ・ユニット・コマンド処理
965:  */
966: int bios_ssue1()
967: {
968:     ssue1(); /* スタート/ストップ・ユニット・コマンド設定処理 */
969:     scsi_bios(); /* μPD72111制御処理 */
970:     return((int)sste);
971: }
972:
973: /*
974:  * BIOSスタート・ユニット・コマンド処理
975:  */
976: int bios_turc()
977: {
978:     turc0(); /* テスト・ユニット・レディ・コマンド設定処理 */
979:     scsi_bios(); /* μPD72111制御処理 */
980:     return((int)sste);
981: }
982:
983: /*
984:  * BIOSモード・エクステンディッド・コマンド処理
985:  */
986: int bios_rec()
987: {
988:     rec0(); /* リード・エクステンディッド・コマンド設定処理 */
989:     return((int)sste);
990: }
991:
992: /*
993:  * BIOSモード・エクステンディッド・コマンド処理
994:  */
995: int bios_wec()
996: {
997:     wec0(); /* ライト・エクステンディッド・コマンド設定処理 */
998:     return((int)sste);
999: }

```

```

1000: /*
1001:  * BIOSモード・エクステンディッド・コマンド処理
1002:  */
1003: int bios_wec1()
1004: {
1005:     wec1(); /* ライト・エクステンディッド・コマンド設定処理 */
1006:     return((int)sste);
1007: }
1008:
1009: /*
1010:  * BIOSモード・ペリフアイ・コマンド処理
1011:  */
1012: int bios_wavc()
1013: {
1014:     wavc0(); /* ライト・ペリフアイ・コマンド設定処理 */
1015:     return((int)sste);
1016: }
1017:
1018: /*
1019:  * BIOSモード・ペリフアイ・コマンド処理
1020:  */
1021: int bios_wavc1()
1022: {
1023:     wavc1(); /* ライト・ペリフアイ・コマンド設定処理 */
1024:     return((int)sste);
1025: }
1026:
1027: /*
1028:  * BIOSモード・ペリフアイ・コマンド処理
1029:  */
1030: int bios_wavc2()
1031: {
1032:     wavc2(); /* ライト・ペリフアイ・コマンド設定処理 */
1033:     return((int)sste);
1034: }
1035:
1036: /*
1037:  * BIOSモード・ペリフアイ・コマンド処理
1038:  */
1039: int bios_wavc3()
1040: {
1041:     wavc3(); /* ライト・ペリフアイ・コマンド設定処理 */
1042:     return((int)sste);
1043: }
1044:
1045: /*
1046:  * BIOSモード・ペリフアイ・コマンド処理
1047:  */
1048: int bios_wavc4()
1049: {
1050:     wavc4(); /* ライト・ペリフアイ・コマンド設定処理 */
1051:     return((int)sste);
1052: }

```



## 第6章

# 自分で作る SCSI のハード&ソフト

ハードディスク&SCSI はアマチュアでも実現可能

宇野沢成夫/橘家鶴蔵/鈴木 洋

最近パソコンにハードディスクは必需品である、という言葉をよく耳にします。

大多数の方々は、市販されている SCSI あるいは SASI のハードディスク・インターフェースに製品のハードディスクを接続していることでしょう。

それはそれでお金を出しさえすればよいのですが、生産が打ち切られた過去のパソコンや、ハードディスクを接続することを前提としていない機種にハードディスクをつなぐためには自分ですべて作らなければなりません。

世の中に「ハードディスク……」というタイトルがついた本はものすごく多いのですが、どうもみんな MS-DOS の使い方の本ばかりでまったく役に立ちません。

そこで自分でハードディスクのシステムを組み上げようと考えている読者へ少しはお役にたてるかもしれない、というのが今回の話題です。

### ●ハードディスク上のデータ

MS-DOS のファイル管理ディスク・デバイスからのデータの読み書きは数バイトからなるブロック単位で行われます。この単位をセクタと呼び **1 セクタは通常物理的に 256, 512, 1024, 2048 バイトが最小単位**になっています。

MS-DOS ではこれらセクタをいくつかにまとめた **クラスタ**単位で読み書きします。

ファイルはこのクラスタの番地のどこが使われているかを管理するための FAT (File Allocation Table) と呼ばれるテーブルでその場所を管理しています。

この FAT の場所などは、あらかじめ決められていて図 1 のようなレイアウトになっています。

フロッピー・ディスクの場合 **FAT** エリアの大きさなどは FAT の最初のエントリに書いてある **メディア・ディスクリプタ**を読み、ドライブ中にあらかじめ用意しておいた **BPB** によって決定するのですが、MS-DOS V3.1 以降のハードディスクの場合は通常 **論理セクタ 0 の予約領域**へ BPB のパラメータを書いておき、それをドライブが利用します (図 2)。

FAT のひとつのエントリにはクラスタの番地が書

かれており、エントリが 16 ビットの場合には最大 64K 番地を指定することが可能です。1 クラスタが 512 バイトの場合  $512 \times 64K = \text{約 } 33M \text{ バイト}$ 、1 クラスタが 2048 バイトの場合で約 134M バイトの容量までアクセスすることができます。

しかし、これを超える 600M バイトや 1G バイト・クラスのドライブが発売されるようになりました。このような場合には、1 台のドライブをいくつかの **MS-DOS でアクセスできる範囲の容量の区画 (パーティション)**に分け、それぞれを論理ドライブとして使うしかありません。

1 台のディスクをいくつかに分けるのですから、その場所がどこからどこまでなのかを管理するテーブルが必要になってきます。これが **パーティション・テーブル**です。

また **1 台のディスクを複数の OS で共有すること**などにも、この方法が用いられます。普通パーティション・テーブルは **SCSI の論理ブロック・アドレスの最初**のほうにあります。管理情報としては **区画の始まりのブロック・アドレス**、**区画のブロック数**、**OS の種類**などが書かれています。図 3 は IBM-PC などで行われているパーティション・テーブルです。

しかし、パーティションについては機種によってサポートされている OS が違うことから統一された仕様というものがありません。

これはハードディスクのように取り外しを前提としていないものではあまり問題にならないのですが、光磁気ディスクのように他の機種へ大量のデータを持っていく場合などに問題になります。この問題は現在解決されていません。

### フォーマット

ハードディスクを使うためにはさきほど説明したように MS-DOS などの **OS で使える構造**にする必要があります。また、ドライブ+ハードディスク・コントローラ (HDC) といった構成のシステムでは **HDC で使えるようにドライブをフォーマット**する必要があります。

ます。前者を論理フォーマットを行うといい、後者を物理フォーマットを行うといいます。

物理フォーマットではディスクの表面をセクタ単位に分け、HDCで読める状態にするのが主な目的ですが、それ以外にもいくつか目的があります。

まず、通常工場から出荷されたドライブには少なからずバッド・スポット(媒体欠陥)が存在します。この部分へデータを書いた場合、データが正常に読み出せなくなる可能性が非常に高くなります。

そこで、物理フォーマット時にその位置を避けるように交代セクタあるいは交代トラックなどのディフェクト処理(代替処理)を行います。これによって、ドライブをアクセスする側からはあたかも連続して正常なセクタが連続してあるように見せることができます。

また、ディスクのインターリーブを変更するのも、この物理フォーマットによって行います。

ただ、最近のコントローラを内蔵した SCSI インターフェースのディスク・ドライブでは、ほとんどトラ

ック単位のバッファを内蔵しているのでインターリーブの変更は逆にスピードの低下をまねくうえ、ディフェクトに関しては工場出荷時点の検査ですでに交替処理は行われているため物理フォーマットは行う必要がなくなってきました。

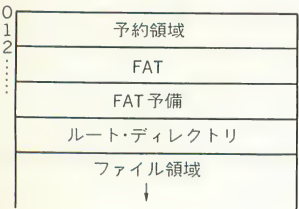
さらに、ディスク・ドライブの構造もハードセクタを採用してセクタ・サイズの変更ができない物が増えており、SCSI でもこの物理フォーマットを行うコマンドの種類はパラメータが微妙に違うので現在ではユーザが物理フォーマットを行わない方向へ向かいつつあるようです。

もし物理フォーマットを行うのであれば、ディスクが十分暖まった後に行うようにしてください。

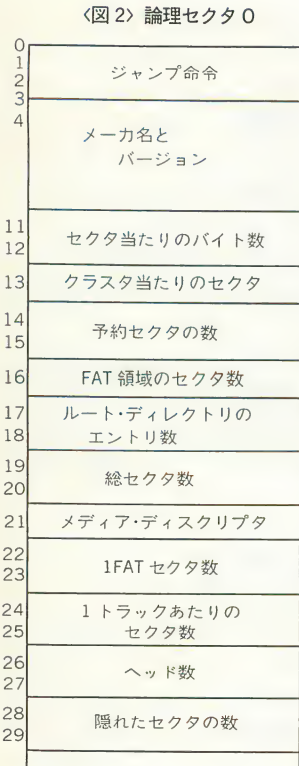
これは物理フォーマットだけに限ったことではないのですが、自動車のエンジンをかけていきなりアクセルを全開にすると故障の原因や寿命が縮まると同様にハードディスクにも暖気運転は必要です。

論理フォーマットではMS-DOSなどのOSで使用されるファイル構造の初期化やパーテーション・セクタなどのディスクのシステム・エリアを初期化します。

OS-9の場合にはファイル構造の初期化に使用されるフォーマット・コマンドはRAMディスクであろう



〈図1〉  
MS-DOS 論理  
レイアウト



〈図2〉 論理セクタ0

〈図3〉 IBM-PC のパーテーション・テーブル

オフセット	先頭からの大きさ(バイト)	内 容
01BEH	16	Partition # 4
01CEH	16	Partition # 3
01DEH	16	Partition # 2
01EEH	16	Partition # 1
01FEH	2	Signature:0AA55H

注) [01FEH]=055H, [01FFH]=0AAH

(a) テーブルの大きさ

コード	意 味
00H	UnKnown
01H	MS-DOS, 12ビット FAT
02H	XENIX
04H	MS-DOS, 16ビット FAT
06H	DOS4.0
11H	東芝日本語 DOS

(c)(b)の04H(システム・インジケータ)の内容

オフセット	先頭からの大きさ(バイト)	内 容
00H	1	Boot indicator
01H	1	Beginning head
02H	1	Beginning sector
03H	1	Beginning cylinder
04H	1	System indicator
05H	1	Ending head
06H	1	Ending sector
07H	1	Ending cylinder
08H	4	Starting sector (relative to beginning of disk)
0CH	4	Number of sectors in partition

(b) テーブルの内容



とハードディスクであろうとひとつで済むのですが、**MS-DOS**ではシステム・ディスクに付属してくるフォーマット・コマンドはハードディスクのブロック型デバイス・ドライバを書いた場合使えないので、このコマンドを書かなければなりません。

このMS-DOSの場合、フォーマット・コマンドはFATエリアの初期化、ルート・ディレクトリの作成、論理ブロック0にある予約領域へBPBの内容を書き込むといった作業を行います。

また、もし1台のドライブを複数の論理ドライブに分けて使うような仕様にするのであれば、**パーティション・セクタを作成するプログラムも別に書かなければなりません**。

## MS-DOSのハードディスク・デバイス・ドライバ

MS-DOSで新しいデバイスを使うときにはデバイス・ドライバをCONFIG.SYSに登録することが必要です。MS-DOSではハードディスクのドライバはブロック型デバイスに分類されるので、これを書けば使えるようになります。

ブロック型のデバイス・ドライバの場合MS-DOS側でファイル構造の管理などを行ってくれるので、要求されるコマンドを逐次処理していけばよいえ、SCSIのハードディスクの場合フロッピー・ディスクとは違い、**要求されるセクタ番号などをトラック/ヘッドと分解して制御する必要がないので、非常に簡単に作ることができます**。

さらに、MS-DOSはシングル・タスクのDOSなので、他のプロセスを気にせずにポーリングもできるのでプログラムはかなり手を抜いて作ることも可能です。

MS-DOSのドライバの仕様はMS-DOSのテクニカル・マニュアルに出ているので詳しく説明しませんが、SCSIのハードディスクのドライバとして必要になる部分をいくつか書いてみます。

### ▶イニシャライズ

ドライバのイニシャライズ・コマンドの部分ではCONFIG.SYSに記述されているパラメータを読み込みます。ドライバの仕様にもよるのですが、接続するドライブのSCSI IDやLUNなどをドライバのパラメータとして与えます。当然この部分でインターフェース・チップの初期化、あるいはワーク・エリアの初期化も行います。

SCSIのドライブではそのSCSI IDに接続されているデバイスが本当にディスクであるかチェックするためにSCSIのインクワイアリ・コマンドを発行してチェックするのもよいでしょう。

よく、この部分でSCSIバスにリセットを送るドラ

イバがありますが、**SCSIの場合安易にバス・リセットをすべきではありません**。なぜなら、複数の機種がSCSIバス上で接続されている場合、それらの機種すべてにリセットが送られてしまうからです。

もちろん、1台しかつながっていなければバス・リセットを送ってもかまわないのですが、通常**SCSIのインターフェース・チップはRSTがアサートされると割り込みがかかるような構造になっているので、そのトラップ・ハンドラを先に仕掛けておかなければなりません**。

SCSIのハードディスクの場合、このRSTがアサートされた後、あるいは電源投入直後に発行されたコマンドのステータス・フェーズは必ずユニット・アテンション(O6H)になります。

そのためこれをクリアするにダミーのリクエスト・センス・コマンドを発行します。

ハードディスクの場合、この部分で物理セクタのサイズを取得したり、パーティション・テーブルを読み込むといったことを行ってもよいのですが、メディアの取り外しのできる光磁気ディスクなどの**リムーバブル・ディスクなどはこのルーチンはドライバがインストールされたときに一度しか呼ばれないのでこの時点ではできません**。ですから最初のアクセスかどうかのフラグを設けておきBUILD BPBルーチンなどで行うべきでしょう。

### ▶メディア・チェック

CCS準拠のドライブの場合、リムーバブルなメディアであるかどうかはインクワイアリ・コマンドで行うことができます。

このルーチンはリード/ライトなどの前に呼ばれるので、リクエスト・センス・コマンドなど無難なコマンドを発行することで、もしメディアが交換された場合にはステータス・フェーズでユニット・アテンション(O6H)が報告され、以前入っていたディスクと違うものであることがわかります。

### ▶BUILD BPB

もし簡単に済ませるのならドライバ内にテーブルを用意しておいてもよいのですが、論理フォーマットのところでも説明したようにMS-DOSの論理セクタ0の部分に書いてあるのであればそれを読んできて返すことで実現できます。

また、このルーチンはドライブ番号を与えられて初めてアクセスするときには必ず呼ばれるので、ドライブをいくつかの区画に分けて使う場合パーティション・テーブルから実際の物理セクタ番号を計算するためのオフセットなどをワーク・エリアへセットする操作などもここで行います。

### ▶オープン/クローズ

MS-DOS V3.1以降でサポートされたファンクショ

ンでデバイス・ヘッダのオープン/クローズ/リムーバブル・メディアのビットを立てた場合にアプリケーションがファイルをオープンあるいはクローズのシステム・コールを実行したときに呼ばれます。

普通、メディアの取り外しができないハードディスクなら無視してもかまわないのですが、光磁気ディスクなどメディアが交換できるドライブではこのファンクションを使ってメディアの取り外しができないようにするために使います。

具体的にはオープン・ファンクションが呼ばれるたびに値が増えるカウンタをワーク・エリアに用意しておき、オープン・ファンクションが呼ばれたときにリムーバブル・メディアでかつオープンの回数がゼロのときドアをロックしてメディアをイジェクトできない

〈図4〉 インクワイアリ・コマンドとインクワイアリ・データ

ビット バイト	7	6	5	4	3	2	1	0
0	オペ・コード (12H)							
1	論理ユニット 番号			保留				
2	保留							
3	保留							
4	割り当て長							
5	ベンダ・ ユニーク			保留			フラ グ	リン ク

- (1) 保留の部分は、通常 0  
(2) フラグとリンクは、コマンドを連結して実行するときに使用。単独で実行するときは、フラグ=0、リンク=0

(a) インクワイアリ・コマンド

ビット バイト	7	6	5	4	3	2	1	0
0	周辺装置機種							
1	RMB	機種限定						
2	ISO バ ージョン		ECMA バージョン			ANSI バージョン		
3	保留							
4	追加長 (n)							
5 n+4								

ベンダ・ユニーク・  
パラメータ  
↓  
(追加長に示さ  
れるバイト数  
のデータ)

#### ・周辺装置機種

00H 直接アクセス装置 (例:磁気ディスク)  
01H 順次アクセス装置 (例:磁気テープ)  
02H プリンタ  
03H プロセッサ  
04H 単一書き込み多重読み取り装置 (例:1 回書き込みの光ディスク)  
05H 読み取り専用直接アクセス装置 (例:光ディスク)  
06H~7EH 保留  
7FH 論理ユニットが存在しない。  
80H~FFH ベンダ・ユニーク

(b) インクワイアリ・データ

ようにします。

クローズ・ファンクションではこの逆にカウンタ値をまず減らし、ゼロならドア・ロックを解除します。

書き込みオープンした状態でメディアを取り出した場合普通はファイルがクラッシュするので、もしリムーバブルなディスクを使用するのであれば積極的に利用したいファンクションです。

#### ▶ リムーバブル・メディア

これも MS-DOS V3.1 以降にサポートされたファンクションです。オープン/クローズ/リムーバブル・メディアのビットが立てられているときに呼ばれます。メディアがリムーバブルかどうかはすでにイニシャライズ・ルーチンで SCSI のインクワイアリ・コマンドで判明しているので単にその結果を返すだけです。

#### ▶ リード/ライト/ベリファイ付きライト

実際の読み書きを行うルーチンです。このルーチンでいかに速く転送を行うかによってディスクのスピードが決まってしまう。

マルチ・タスクの OS などでは本来 DMA 転送を行うべきなのでしょうが、シングル・タスクの MS-DOS ではあまりそのメリットがあるとはいえません。パラメータとして渡されるのは論理セクタ・アドレスですからこれをドライブの物理セクタ・アドレスに変換し、パーティションを切っている場合にはそのオフセットを加えます。

また、データの信頼性については、ハードディスク

〈図5〉 DTC-310DB の SASI コマンド

		SC601	DTC 系	XEBEC 系
コマンド		C2H	C2H	OCH
パ ラ メ ー タ	0	デバイス・モード	01H	シリンダ数(MSB)
	1	00H	ステップ周期	シリンダ数(LSB)
	2	00H	ステップ・モード	ヘッド数
	3	00H	ヘッド数	RWC(MSB)
	4	00H	シリンダ数(MSB)	RWC(LSB)
	5	ヘッド数	シリンダ数(LSB)	WPC(MSB)
	6	シリンダ数(MSB)	RWC(LSB)	WPC(LSB)
	7	シリンダ数(LSB)	RWC(MSB)	ECCバースト・ビット
	8	RWC(MSB)	00H	—
	9	RWC(LSB)	00H	—
	10	WPC(MSB)	—	—
	12	WPC(LSB)	—	—

RWC: Reduce Write Current Cylinder

WPC: Write Precompensation Cylinder

#### SC601 デバイス・モード

ビット 2: "0" DK-511 シリーズ, "1" その他  
ビット 0: "0" 256 バイト/s, "1" 512 バイト/s  
他のビットはすべて "0"



の場合 ECC によるエラー・コレクションが行われるので通常転送されてくるデータが化けてくることはまずあり得ません。もし、エラーが起こり ECC によって修正された場合はステータス・フェーズでリカバリ・エラー (01H) になります。この場合完全に読み出すことができなくなる前に代替処理を行うべきでしょう。ECC でも修正できなかった場合には通常はデータが送られてきません。

ベリファイについては SCSI のコマンドのベリファイを用いて行います。ドライブによっては書き込んだデータがトラック・バッファに蓄えてあるので**すぐに同じアドレスから読み込んでベリファイを行う方法ではデータをメディアから読み込むのではなくメモリにあるデータを転送してくる場合もあるので、あまり意味がありません。**

ドライブを書く上で必ず使う SCSI のコマンドは、いうまでもなくリード (08H) とライト (0AH) ですが、それ以外にもドライブの機能として必要なものがいくつかあります。

まず、OS にもよるのですが、ドライブの初期化ルーチンで接続されているドライブの物理ブロック・サイズなどある程度の仕様を知る必要があります。SCSI のドライブではリード・キャパシティ (25H) によって物理ブロック・サイズと最大ブロック・アドレスを知ることができます。

また、光磁気ディスクなどのリムーバブル・メディアを使用する可能性がある場合や SCSI バスに接続された、使用しようとするデバイスが本当にハードディスクなどのダイレクト・デバイスであるか、といったチェックにはインクワイアリ (12H) というコマンドを

使用します (図 4)。

リムーバブル・ディスクの場合は、通常メディアの取り出しを禁止するためのコマンド PREVENT/ALLOW MEDIUM REMOVAL (1EH) がサポートされているので、さきのオープン/クローズなどで利用します。

一般的に SASI のドライブではこれらのコマンドはサポートされていないため、**前もって物理ブロックを使用できるサイズにしておくなどの対策が必要です。**

その他 SASI の場合、ドライブを最初にアクセスするときにドライブの仕様を知らせる必要がある場合もあります。

具体的にはアサイン・ディスク・パラメータという名前のコマンドを発行します。

SCSI の場合、通常はドライブの記録面の数シリンダ分をシステム領域としてそのメディアに仕様やディフェクトなどの情報を記録しておくようになっていますが、それより前に製品として登場した SASI ではそのような処理は行われず、ボード上のジャンパの設定やファームウェアの ROM のテーブルなどによりデフォルトのシリンダ数やヘッド数を持っています。

OEM 向けのコントローラ・ボードの場合、ドライブの仕様などは決定されているのでそのデフォルト値でなにも問題なく使えますが、組み合わせるドライブが別のコントローラ・ボードの場合にはそれとは違った仕様のドライブを接続する可能性のほうがあるくらいです。このコマンドはコントローラ・ボードによりコマンド・コードやパラメータがまちまちです。

代表的な SASI のハードディスク・コントローラ・ボードのコマンドは図 5 のようになっています。

〈図 6〉 センス・データ

ビット バイト	7	6	5	4	3	2	1	0
0	AdValid	誤り区分			誤りコード			
1	製造者規定			論理ブロック・アドレス(MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス(LSB)							

(a) 非拡張センス・データ形式

ビット バイト	7	6	5	4	3	2	1	0
0	AdValid	誤り区分			誤りコード			
1	セグメント番号							
2	ファイル・マーク	EOM	ILI	保 留	センス・キー			
3	情報バイト (MSB)							
4	情報バイト							
5	情報バイト							
6	情報バイト (LSB)							
7	追加センス長							
8-n+7	追加センス・バイト							

(b) 拡張センス・データ形式

ステータス・フェーズで 00H(ノー・センス)以外の場合はなんらかの問題があったときです。このときドライブではリクエスト・センス(03H)コマンドでこれをクリアすると同時に原因を探り、適切なエラーを返すのですが、このコマンドによって返されるデータは図 6 のように 2 種類あります。

現在発売されている製品では SCSI のドライブではエラー・クラス 7 の拡張センス・データ形式、SASI のドライブでは 4 バイトの非拡張センス・データ形式を返すようです。この**非拡張センス・データ形式はドライブ(コントローラ)によってエラー・コードがまちまち**なので、ドライブはエラー時の処理をその原因を作ったコマンドによって判断する方法をとるほうが無難です。

## SCSI/SASI のコントローラ・ボード

最近のドライブは SCSI のコントローラ内蔵のものが主流になってきているのでもはや死語になった感があるのですが、ディスク・ドライブ+ハードディスク・コントローラの組み合わせのディスク・システムは最近リース期限が切れて秋葉原などでジャンクとしてかなり出まわるようになってきました。

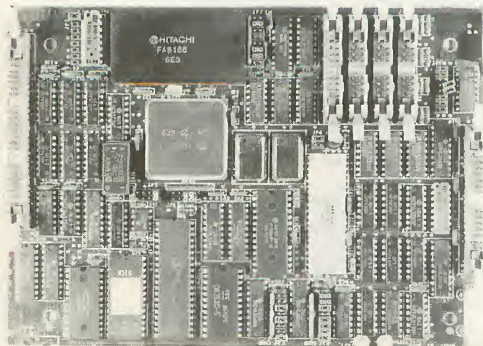
当時は 20 Mバイトあたりが主流だったので現在の肥大化したアプリケーションではとても容量が足りないうえにスピードも現在のものに比べかなり遅く感じます。

ところが、値段は**フロッピー・ディスクでその容量分を買うよりも安い場合があるので(ジャンクですから当然リスクを考えなければなりません)**バックアップ用として利用するにはよいかもしれません。

そこで、簡単ですが秋葉原などに出来る可能性のあるハードディスク・コントローラをいくつか紹介してみます。

### ▶ SC601-1/SC601-2/SC601-3(写真 1)

日立の SASI-ST506 の MFM のコントローラです。



〈写真 1〉 SC601-1

4 台までのドライブを接続することができトラック・バッファを内蔵しているので、比較的高速で転送が行えます。

通常このコントローラは同社の 5 インチのドライブである DK511 シリーズとの組み合わせで使われる設定が施されており、他のドライブで使用する場合はジャンパの設定を変更する必要があります。

変わったところでは SC601-1 が同社の ST506 インターフェースのフロッピー・ディスク・ドライブ FDD-411(8 インチ・フォーマット容量 6.15M バイト)および SC601-2 および SC601-3 が FDD-451(5 インチ・フォーマット容量 4.15M バイト)に接続できます。

### ▶ DS300/DS500

NEC の SASI-ST506 の MFM のコントローラです。同社のパソコン PC9801 シリーズの普及により国内ではこのコントローラの工場出荷時の設定コマンド体系がかなり一般的なものになっています。

DS300 は 3.5 インチ・サイズで 8 ヘッドまで、DS500 が 5 インチ・サイズで 16 ヘッドまでのドライブが制御可能です。

これらのコントローラはディップ・スイッチの設定によって D モード/X モード/S モードの 3 種類のコマンド体系が選択できるようになっています。

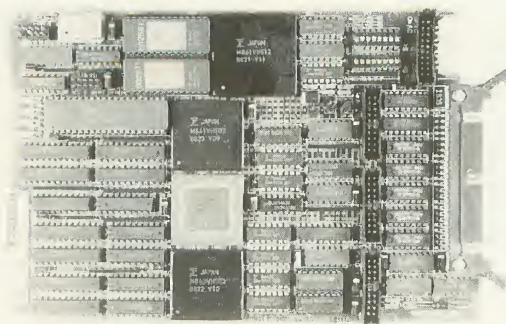
それぞれアメリカの有名なコントローラ・メーカーの頭文字をとったネーミングのようで、だいたいどのメーカーのコマンドとコンパチか想像がつくと思います。

工場出荷時は D モードに設定されており、サード・パーティの PC9801 シリーズ用 SASI のハードディスクではこれに習ったコマンド体系になっているようです。

### ▶ M1055B

富士通の SCSI-ST506 の MFM のコントローラです。初期の FM-R シリーズのハードディスクで使われていたコントローラで、コマンド体系は CCS の規格をほとんど満たしています。

このコントローラは NCL 社より OEM 供給されていたもので NCL 社では NDC-3016 という型番で呼ば



〈写真 2〉 M1054A



〈表 1〉  
SPCの種類

	MB87030	MB87031	MB87033	MB89351	MB89352
ドライバ/レシーバ	外付け	外付け	内蔵	外付け	内蔵
同期転送	○	○	○	×	×
転送カウンタ	24 ビット	24 ビット	28 ビット	24 ビット	24 ビット
パッケージ	88 ピン PGA	100 ピン QFP	80 ピン QFP	64 ピン QFP 64 ピン SH-DIP	48 ピン QFP 48 ピン DIP

れています。

▶ M1054A(写真 2)

富士通の SCSI-ESDI のコントローラです。一応 SCSI の規格を満たしてはいるものの、CCS のコマンド体系ではありません。

接続可能ドライブは同社 M2244CA/M2245CA/M2246CA の 3 機種に限られているようです。

▶ ACB4000/ACB4070

米アダプテック社の SCSI-ST506 のコントローラです。記録方式は ACB4000 が MFM, ACB4070 が RLL です。

このコントローラはある程度 CCS コマンド体系をサポートしているものの、フォーマットやディフェクト処理などのコマンドについては CCS と違っているのが注意が必要です。

アメリカではマッキントッシュなどのハードディスクでかなり使われているらしく、かなりメジャーなコントローラの様です。

▶ ACB4520

米アダプテック社の SCSI-ESDI のコントローラで先の ACB4070 などより後から設計されたものらしく、コマンド体系は CCS に完全に対応しています。

▶ M1052A/M1052B(写真 3)

富士通の MDC-SA4000 のコントローラです。SCSI インターフェースではないので注意が必要です。

このコントローラは富士通の FM11 や FM16 $\beta$  などのハードディスク用として使われていたもので、M1052A が 20M バイトまでのステップング・モータを使用している SA4000 バスのドライブを、M1052B はそれらを含み 67M バイトまでのものをコントロール

することができます。

SA4000 のドライブというのは富士通のドライブの型番でいうと M2235B, M2243B といったラスト・レターが B のもので、コネクタが 50 ピンのカード・エッジになっているものです。なお、MDC のコネクタも同じ 50 ピンなので、SCSI と勘違いして買ってしまいそうですが、これらを SCSI のインターフェースに接続しても何があろうと動きません。

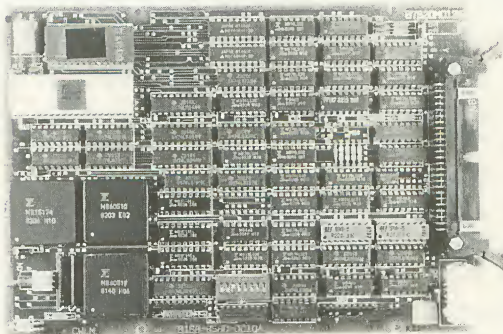
## SCSI インターフェース

SCSI のコントローラ・チップはいくつか発売されていますが秋葉原などで比較的入手しやすいチップは富士通の MB89352 と NCR 社の NCR5380 です。

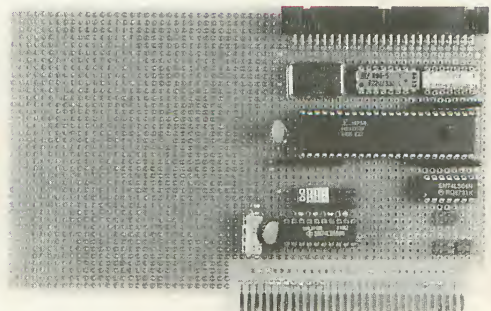
NCR5380 はかなり前から発売されていたのであって、いろいろな記事に紹介されているので説明の必要はないでしょう。NCR5380 の場合 SCSI のフェーズをソフトウェアで実行しなければならず、SCSI のプロトコルを理解するにはよいのですが、パフォーマンスは若干落ちます。

MB89352 は MB87030 という同社の SCSI コントローラ・チップから、同期転送機能を外し SCSI の不平衡ドライバを内蔵したもので、最近では J3100 シリーズや X68000 シリーズの SCSI インターフェース・カードにも搭載されるようになり、かなりメジャーなチップになってきています。

特徴はデータ転送時にデータ・バッファに 8 バイトの FIFO バッファと 24 ビット・カウンタを持っているため転送を高速に行えることです。なお、富士通ではこの SCSI コントローラを SPC(SCSI PROTOCOL

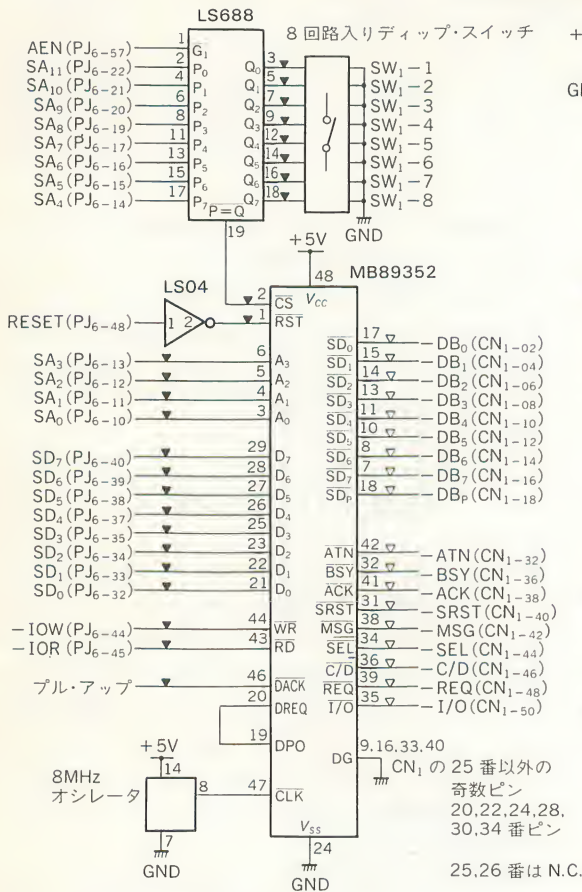


〈写真 3〉 M1052A



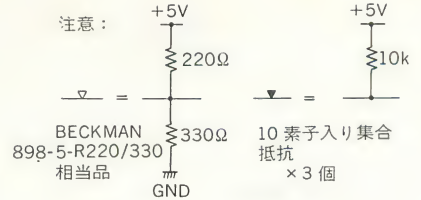
〈写真 4〉 SPC ボード

〈図7〉 J3100用SPCボード



+5V(PJ<sub>6-02</sub>)  
(PJ<sub>6-52</sub>)  
GND(PJ<sub>6-01</sub>)  
(PJ<sub>6-09</sub>)  
(PJ<sub>6-18</sub>)  
(PJ<sub>6-27</sub>)  
(PJ<sub>6-43</sub>)  
(PJ<sub>6-51</sub>)  
(PJ<sub>6-60</sub>)

注意:



▶ 信号線とコネクタの対応を書いておきます。

AEN(A <sub>11</sub> )	SA <sub>11</sub> (A <sub>20</sub> )	SA <sub>10</sub> (A <sub>21</sub> )	SA <sub>9</sub> (A <sub>22</sub> )	SA <sub>8</sub> (A <sub>23</sub> )
SA <sub>7</sub> (A <sub>24</sub> )	SA <sub>6</sub> (A <sub>25</sub> )	SA <sub>5</sub> (A <sub>26</sub> )	SA <sub>4</sub> (A <sub>27</sub> )	SA <sub>3</sub> (A <sub>28</sub> )
SA <sub>2</sub> (A <sub>29</sub> )	SA <sub>1</sub> (A <sub>30</sub> )	SA <sub>0</sub> (A <sub>31</sub> )	-IOW(B <sub>13</sub> )	-IOR(B <sub>14</sub> )
SD <sub>7</sub> (A <sub>2</sub> )	SD <sub>6</sub> (A <sub>3</sub> )	SD <sub>5</sub> (A <sub>4</sub> )	SD <sub>4</sub> (A <sub>5</sub> )	SD <sub>3</sub> (A <sub>6</sub> )
SD <sub>2</sub> (A <sub>7</sub> )	SD <sub>1</sub> (A <sub>8</sub> )	SD <sub>0</sub> (A <sub>9</sub> )	RESET(B <sub>2</sub> )	

+5V(B<sub>3</sub>, B<sub>29</sub>) GND(B<sub>1</sub>, B<sub>10</sub>, B<sub>31</sub>)

ディップ・スイッチの1~8が、I/OアドレスのA<sub>11</sub>~A<sub>4</sub>に相当しています。Onでそのアドレスが00ffで1になります。

ですから、例えばアドレスを0300Hに設定するには、1番から順番に On On Off Off On On On On とディップ・スイッチを設定します。

▶ 回路がちゃんとつながっているのに動かない、という場合、半分くらいは電源、グラウンドの配線が弱いからです。まず最初に部品の配置を決めて、その後グラウンド、電源を強力に配線して、その後信号線を配線するようにしましょう。

私は通常は「はんだ吸い取り線」を使ってグラウンド、電源を配線しています。また、パソコンではICのすぐ脇に実装する事も忘れずに！今回の場合、終端抵抗にもパソコンを入れた方が良いと思います。

CONTROLLER)と呼んでおり、シリーズにはこのMB89352の他に表1のような種類があります。

### ● 実際の回路

MB89352の場合、いわゆる80バスの信号インターフェースによって制御されるので、86のバスを持つパソコンではチップ・セレクト信号を作るくらいで非常に簡単にSCSIインターフェース・カードができてしまいます。

ただ、それ以外のバス信号を持つパソコンの場合には若干バス・タイミングの修正などが必要なので外付け部品が必要になります。

また、DMA転送を行う場合リクエストをレベル・センスによって転送を行うDMAコントローラでは問題ないのですが、エッジ・トリガが必要なものでは若干の工夫が必要になります。

いくつかのパソコンで接続する場合の回路図を図7~図9に示します。また、製作したIBM用SPCボードを写真4に示します。

### ● SPCのドライバ

対象とするシステムはMS-DOS V3.1以降です。機種は問いません。リスト1~リスト7に全プログラ

ム(ドライバ、フォーマット・プログラム)を示します。MAKEFILEを実行することで.SYSと.EXEができます。

ただし、FATの大きさを4085以下にすればV2.1でも動作します。

8086/186コードで書かれています。使用したアセンブラはMASM4.0/5.1です。

DMAを使用していないので、マシンのハードにほとんど影響されません。MS-DOSマシンならばどれでもI/Oアドレスの変更だけで動くはずですが。

現在のところ、

▶ J3100SGT/GX/SL/SS(英語DOS3.3/日本語DOS3.1)

▶ PC9801VX41(MS-DOS3.3)

▶ IBM-PC/XT, JX, 5550(DOS3.3)

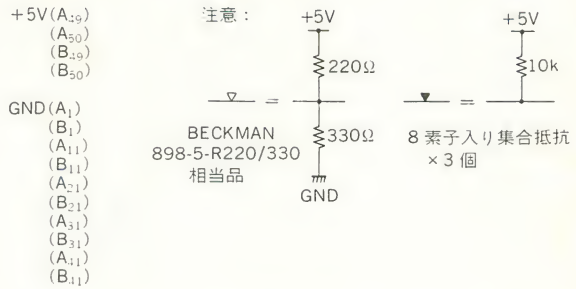
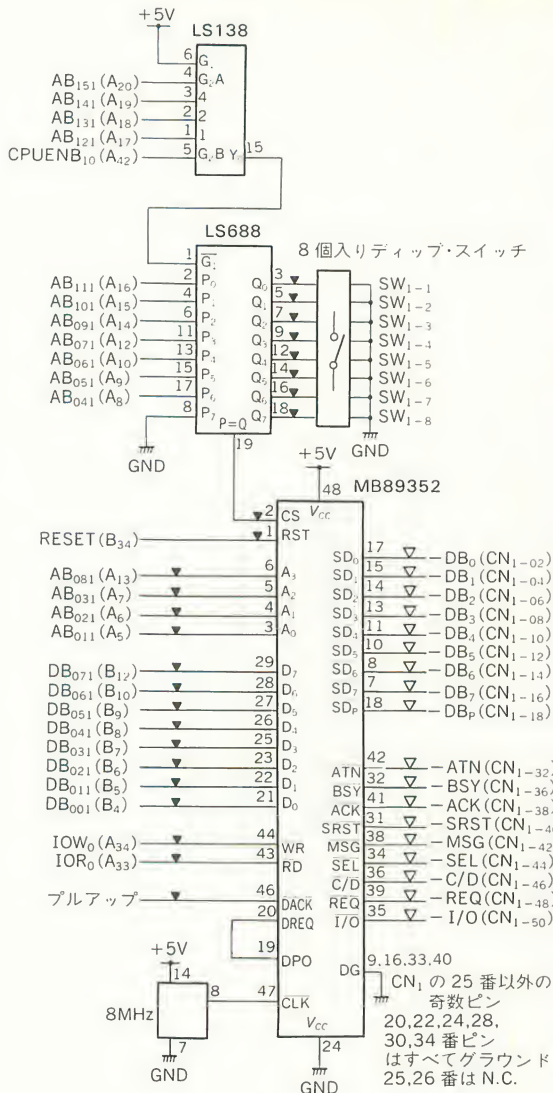
▶ FM11AD2+(MS-DOS2.1/3.1)

で動作確認されています。どれも、I/Oアドレス以外にはまったく手を加えずに動作しています。また、FM11では現在同一ドライブをパーティションを切つてOS9と共用で使うことすら可能です。

対象になるドライバはANSIの標準に準拠した



〈図8〉 PC9801用 SPC ボード



▶ディップ・スイッチの1〜8がI/OアドレスのA<sub>11</sub>〜A<sub>4</sub>に相当しています。Onでそのアドレスが0, Offで1になります。A<sub>15</sub>〜A<sub>12</sub>は0に固定です。ですから、例えばアドレスを02E0Hに設定するには、1番から順番に On On Off On Off Off On とディップ・スイッチを設定します。

▶PC9801は他の機種と違って8ビット・バスで奇数番地I/Oができません。このため、アドレスは2番地おき、上位8バイトと下位8バイトは100H番地離してロケーションしています(SPCのA<sub>3</sub>に、バスのA<sub>8</sub>が入っている)。また、I/Oアドレスはフル・デコードしています。それ以外はJ3100用と同じです。製作上の注意については、J3100用回路図を参考にして下さい。

いるのはSONY製が多いようなので**多分動く**と思います。

#### ● ドライバの使用方法

SPCカードのハードウェアができあがったらカードのI/Oアドレスを空いているアドレスに設定し、SFFORM.EXEを実行してください。

IBM・PC互換機では0300H、98互換機では**02E0H**あたりがとりあえず適当だと思います。このSPC.SYSでは**0300H**になっています。

SFORMはMS-Cで書いてあり、SCSIのハンドシェイクもすべてCでやっているの**で、実行時にSPC.SYSがある必要はありません。**

なお、SFFORMはPC9801の場合はI/Oアドレスを変更して再コンパイルが必要です(8ビット・バスでの奇数番地のI/Oアクセスができないため)。その他の機種の場合はこのままで実行可能はずです。

これが正常終了したら、ハードウェアは**多分正常**なので、つぎにSPC.INCの中のI/Oアドレスを各自のマシンのアドレスに書き替えて再アSEMBルしてください。

あとはCONFIG.SYSにSPC86.SYSまたはSPC186.SYSを組み込めば終わりです。

CPUが8086・80C86の場合はSPC86.SYSを、V20/30/186/286/386の場合はSPC186.SYSを使ってください。SPC86.SYSは当然186や286でも動きますが、スピードが10%程度は落ちます。

なお、起動時に論理ブロック0を読み込んでパーテーション情報などをセットしているので、起動時には

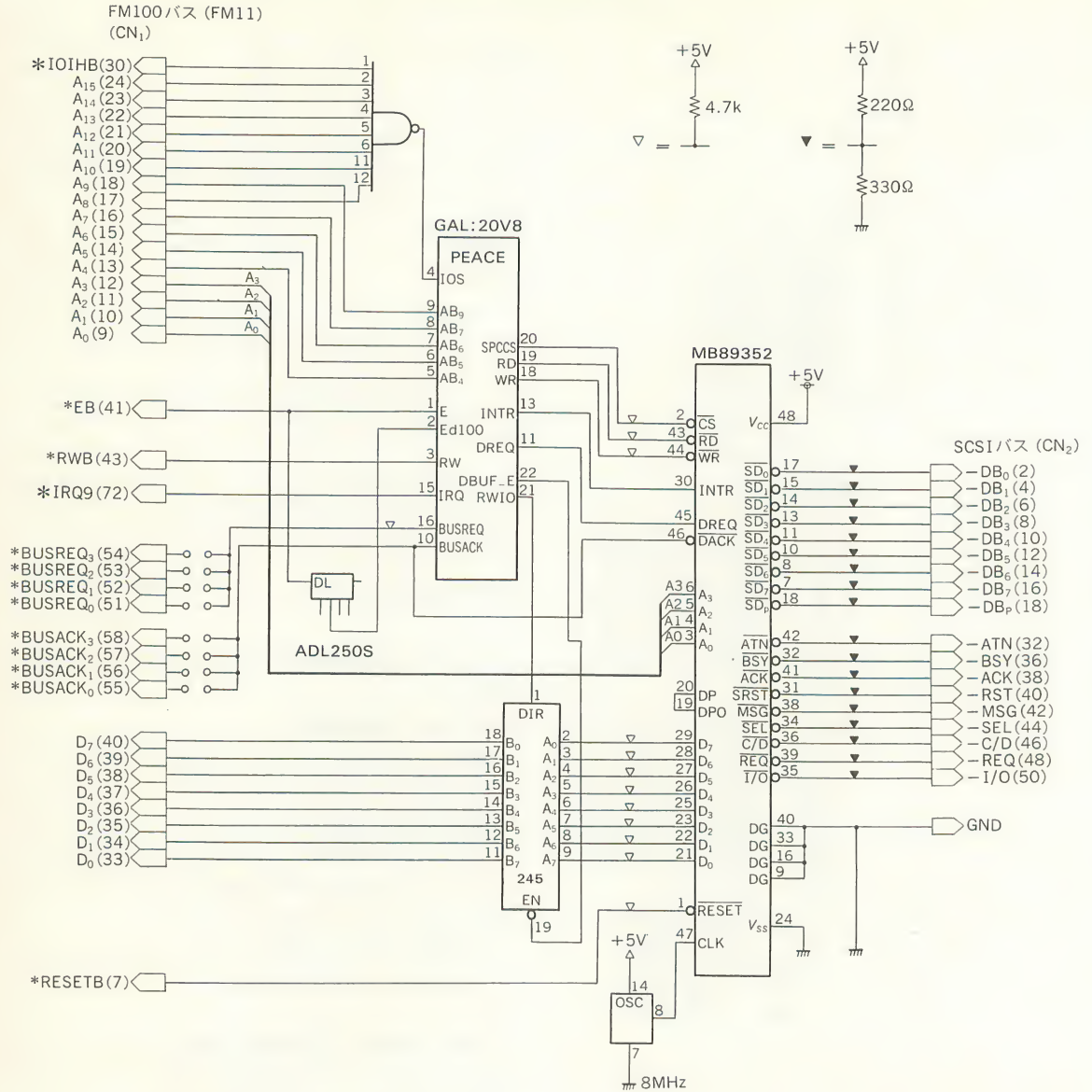
SCSIドライブ(CCSが使えるドライブ)、SONYおよびそのOEMのMOディスクです。

具体的にはNEC PC9801-55カード(あるいは同等品)に接続できるドライブ、および富士通FM-R/TOWNSシリーズに接続できるドライブなどであればまったく問題なく使用できます。

PC9801-27カードに接続できるドライブについては、イニシャライズ・ルーチンとステータス・チェック・ルーチンを書き替えれば動作可能ですが、このままでは動かないと思います。特にTDC384などはROMがいろいろあるらしいので……。

Mac用については仕様がいろいろあるそうなので、すべてが変更なしに接続できるという保証はできませんが、動かなくても一部パッチを当てれば動くと思います(ただし未確認)。MOはMac用として売られて

〈図 9-a〉 FM11 用 SPC ボード



〈図 9-b〉 FM11 用 SPC ボードの GAL

<p>ABEL(™) Version 2.10a - Document Generator          SPC Card Decoder &amp; Timing Generator          1989/06/08 Designed by TheEND. Named by AZUKI          Equations for Module _PEACE</p> <p>Device PEACE</p> <p>- Reduced Equations:</p> <pre> ~SPC_CS = !(AB4 &amp; AB5 &amp; !AB6 &amp; AB7 &amp; !AB9 &amp; !~IOS);  ~DBUF_E = !(~BUSACK &amp; !~E # !~BUSACK &amp; !~Ed100 # AB4 &amp; AB5 &amp; !AB6 &amp; AB7 &amp; !AB9 &amp; !~E &amp; !~IOS # AB4 &amp; AB5 &amp; !AB6 &amp; AB7 &amp; !AB9 &amp; !~Ed100 &amp; !~IOS);         </pre>	<p>Page 1          30-Aug-89 01:07 PM</p> <pre> RWIO = (!~BUSACK &amp; !RW # ~BUSACK &amp; RW);  ~RD = !(~BUSACK &amp; !~E &amp; !~Ed100 &amp; !RW # ~BUSACK &amp; !~E &amp; !~Ed100 &amp; RW);  ~WR = !(~BUSACK &amp; !~E &amp; !~Ed100 &amp; RW # ~BUSACK &amp; !~E &amp; !~Ed100 &amp; !RW);  enable ~BUSREQ = (~BUSACK &amp; DREQ);  ~BUSREQ = ! (1);  enable ~IRQ = (INTR);  ~IRQ = ! (1);         </pre>
---	--



必ず HDD がレディ状態になっている必要があります。

MO の場合にはデバイス番号を取得する必要上、ブート時には MO 媒体が入っている状態にしておいてください。入れておかないとイニシャライズ・エラーになります。

CONFIG.SYS にパラメータが入れられるので ID と LUN が設定可能です。

I/O アドレスは固定になっています。ですからこれらを変更したい場合にはソースを直して再アSEMBルする必要があります。

アドレスは SPC.INC で定義しています。現在の I/O アドレスは 0300H 番地から、デフォルトのターゲット ID は 0、LUN は 0 です。

このドライバ単体ではマルチ・ターゲット(複数台の HDD)はサポートしていませんが、CONFIG.SYS で SCSI ID か LUN を変更してドライバを複数回登録すれば複数台接続も可能です。若干メモリの無駄になりますが、1 ドライブあたり 1.5K バイト程度なので我慢できると思います。

CONFIG.SYS の例：

DEVICE=SPC186.SYS /IO /L0

DEVICE=SPC186.SYS /IO /L1

上の例では、ID0、LUN0 と ID0、LUN1 の 2 台の HDD が使えるようになります。

ターゲット ID というのは SCSI コントローラ(ドライバ)の SCSI バス上での ID です。7 はパソコン側が使うので、0 から 6 までです。

LUN というのは、SCSI コントローラとハードディスク・ドライブ間での機番です。ST506 インターフェースの HDD の場合、ドライブの機番が 1 から 4 まで選べますが、これが 1 のときに LUN=0 です。SCSI 内蔵型 HDD の場合は、必ず 0 になります。

パーティションは 12 まで切れます。それぞれ最大 32M バイトまでです。ドライバ・イニシャライズのときにドライブの論理セクタ 0(DD セクタ)を読み込んで、それから判断するので、ブートの際に必ずデバイスがレディになっている必要があります。

MS-DOS には、「標準で組み込まれているデバイスよりも大きなセクタ・サイズのブロック・デバイスは追加できない」という闇の掟があります。このため、IBM-PC などでは 512 バイト/セクタ以上にできないので、32M バイト以上のパーティションが切れません。日本語 DOS しか絶対に使用しないというのならば 1024 バイト/セクタもできるので、64M バイトのパーティションも可能です。

DDセクタの構成はつぎの通りです(論理ブロック0)。

2 バイト：パーティションの個数

4 バイト \* 12：それぞれのパーティションの開始する論理ブロック・アドレス

14 バイト：未使用

16 バイト \* 12：パーティションごとの BPB 情報

合計 256 バイト

DMA 転送はサポートしていませんが、CPU の性能さえ十分であれば通常の HDD よりはるかに高速でアクセスできます。

J3100GX の 286-12MHz 時は実測でデータ・フェーズの転送レートが 560K バイト/s 出ており、PC9801 の DMA 転送時(実測 300K バイト/s)よりも約 1.8 倍高速です。つまりは、286-8MHz 以上ならば 8237 を使った DMA 転送には絶対負けません。V30-8MHz で同等のようです。

以下にベンチ・マーク・テストの結果を示します。

内容：3788800 バイトのファイルの同一 HDD 内でのコピー

J3100GX (80286-12MHz)  
+FMHD-A11 19 秒

J3100GX (80286-12MHz)  
+M1055B+D3126(インターリーブ 2) 44 秒  
PC9801VX41(80286-10MHz)  
+FMHD-A11 18 秒

参考までに、純正 HDD&ドライバでは(すべて DMA 使用)以下の通りの結果でした。

J3100GX (80286-12MHz)  
+内蔵 HDD(コナー 40M バイト) 34 秒  
PC9801VX41(80286-10MHz)  
+内蔵 HDD(D3126) 79 秒  
PC9801VX41(V30-8MHz)  
+内蔵 HDD(D3126) 82 秒  
FMR70HX2(80386-20MHz)  
+FMHD-A11 20 秒

D3126 というのは、NEC の 3.5 インチ HDD(20M バイト)です。FMHD-A11 というのは、富士通製の FM-R シリーズ用外付け HDD(130M バイト)です。M1055B というのは FM-R シリーズの内蔵 HDD や FMHD-411/611 などに使われている SCSI コントローラです。

トラック・バッファがないコントローラの場合、インターリーブ値によって転送速度が大きく変化します。また、256 バイト/セクタの場合 512 バイト/セクタのときに比べて若干データ転送レートが落ちます。つまり、512 でインターリーブ 2 では駄目でも 256 だと間に合う場合もあるわけです。

このあたりの事情は、上の結果を見てもわかる(12MHz の J3100GX よりも、10MHz の 98 の方が速い、しかし内蔵 HDD は J3100 の方が 2 倍も速い)とおり、千差万別なので実機でテストしてみるしかありません。

このドライバはMOに対応していますが、パーティションの切り方を変えた媒体を入れ替えたときに問題が生じます。これはMS-DOSのシステム側の手抜きが原因なのですが、逃げる方法としてはBPBのメディア・ディスクリプタ・バイトをパーティションが違うもの同士でかみ合わないよう変更する以外ありません。

パーティションの切り方を変えた媒体を入れ替えたら、リセットするのが一番簡単ですが、もし「どうしても！」という方がいらっしゃいましたら筆者までご相談ください。

なお、同じ切り方の媒体でしたら、いくら入れ替えても大丈夫です(書き込み中などを除く)。

#### ● SPC のフォーマット・プログラム

SPC カード用物理フォーマット/パーティション・ユーティリティです。

MS-C で書いてあります。I/O の操作もすべてCでやっているのので、使用する際に SPC.SYS が登録されている必要はありません。

I/O アドレス、SCSI ID、LUN は最初に手で入力するようになってるので、PC9801 以外ではそれらを変更して再コンパイルする必要はありません。

媒体検査でエラーになった個所は自動的にディフェクトとして登録され、交替処理が行われます。

ジャンク屋などで買ってきたドライブは初回の物理フォーマット後に媒体検査を多数回(できれば10回位以上)やっておくと、比較的安心して使えます(特に、ディフェクト・リストが付いていなかった場合)。

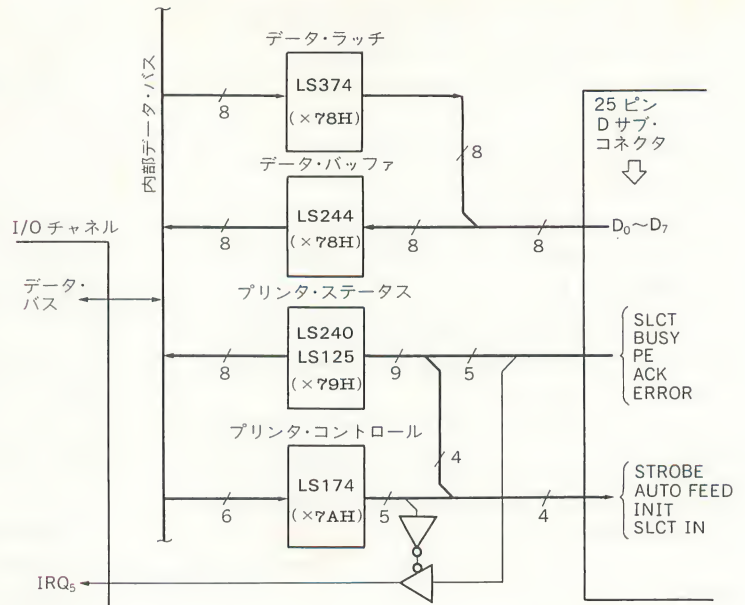
M1055B+D3126 で、初めてフォーマットを行う場合の操作例を以下に示します。ここでは、

▶ SPC カードの I/O アドレスは 0FDFO 番地

▶ セクタ・サイズは 256 バイト/セクタ

▶ SCSI ID、LUN は 0、ディフェ

〈図 10〉 パラレル・インターフェース・ブロック図



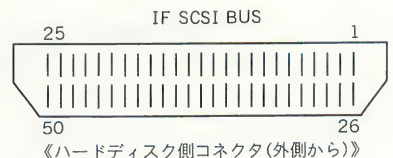
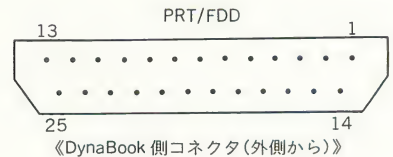
〈図 11〉 Easy Hard 用ケーブル

《DynaBook》	《ハードディスク》
プリンタ・ポート	SCSI バス
- STROBE (1)	(44) - ACK
D <sub>0</sub> (2)	(26) - D <sub>0</sub>
D <sub>1</sub> (3)	(27) - D <sub>1</sub>
D <sub>2</sub> (4)	(28) - D <sub>2</sub>
D <sub>3</sub> (5)	(29) - D <sub>3</sub>
D <sub>4</sub> (6)	(30) - D <sub>4</sub>
D <sub>5</sub> (7)	(31) - D <sub>5</sub>
D <sub>6</sub> (8)	(32) - D <sub>6</sub>
D <sub>7</sub> (9)	(33) - D <sub>7</sub>
- ACK (10)	(49) - REQ
BUSY (11)	(46) - MSG
PE (12)	(43) - BSY
SELCT (13)	(48) - C/D
- AUTFD (14)	(47) - SEL
- ERROR (15)	(50) - I/O
- IPRT (16)	(41) - ATN
- SLIN (17)	(45) - RST
GND (18~25)	(1~12, 14~25) GND
DSUB25J	AMPHENOL50J (シールド)

(a) HDD 側のコネクタがアンフェノール・タイプの場合

《DynaBook》	《ハードディスク》
プリンタ・ポート	SCSI バス
- STROBE (1)	(38) - ACK
D <sub>0</sub> (2)	(02) - D <sub>0</sub>
D <sub>1</sub> (3)	(04) - D <sub>1</sub>
D <sub>2</sub> (4)	(06) - D <sub>2</sub>
D <sub>3</sub> (5)	(08) - D <sub>3</sub>
D <sub>4</sub> (6)	(10) - D <sub>4</sub>
D <sub>5</sub> (7)	(12) - D <sub>5</sub>
D <sub>6</sub> (8)	(14) - D <sub>6</sub>
D <sub>7</sub> (9)	(16) - D <sub>7</sub>
- ACK (10)	(48) - REQ
BUSY (11)	(42) - MSG
PE (12)	(36) - BSY
SELCT (13)	(46) - C/D
- AUTFD (14)	(44) - SEL
- ERROR (15)	(50) - I/O
- IPRT (16)	(32) - ATN
- SLIN (17)	(40) - RST
GND (18~25)	(25 以外の奇数ピン, 20, 22, 24, 28, 30, 34)
DSUB25J	フラット・ケーブル用コネクタ

(b) HDD 側がフラット・ケーブル・タイプの場合





クト個所が2個所

▶ 10M バイトずつふたつのパーテーションに分割  
という操作を行っています。

<CR>が付いている部分が入力部分です。

SPC カードの I/O アドレスはどこになっていますか？ FDF0<CR>

フォーマットしたいドライブの SCSI-ID は何番ですか？ 0<CR>

ロジカル・ユニット番号(ドライブの番号)は何番ですか？ 0<CR>

物理フォーマットを行いますか？(Y/N) Y<CR>

セクタ・サイズをいくつにしますか？(256・512) 256<CR>

インターリーブ・ファクタは、いくつにしますか？ 2<CR>

一度もフォーマットを行ったことがないドライブですか？(Y/N) Y<CR>

このドライブの、シリンダ数はいくつですか？ 615<CR>

ヘッド数はいくつですか？ 4<CR>

書き込み位相補正(ライト・プリコンベ)を始めるシリンダはいくつですか？(不要な場合は、シリンダ数と同じ数) 256<CR>

書き込み電流補正(リデュースド・ライト・カレント)を

#### 〈図 12〉 パラレル・インターフェース・レジスタの割り当て

・データ(×78H または ×7CH)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

ライト:SCSI バス・データ出力(プリンタ出力データ)

リード:SCSI バス・データ入力(プリンタ入力データ)

・プリンタ・コントロール(×7AH または ×7EH)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

未使用

ACK(STROBE)

SEL(AUTO FD XT)

ATN(INIT)

RST(SLCT IN)

IRQ Enable

・プリンタ・ステータス(×79H または ×7DH)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

未使用

I/O(ERROR)

C/D(SLCT)

BSY(PE)

REQ(ACK)

MSG(BUSY)

始めるシリンダはいくつですか？(不要な場合は、シリンダ数と同じ数) 615<CR>

これでよいですか？(Y/N) Y<CR>

ディフェクトの個数はいくつですか？ 2<CR>

ディフェクト 1: シリンダは？ 100<CR>  
ヘッドは？ 0<CR>

インデックスからの距離(バイト)は？ 6535<CR>

ディフェクト 2: シリンダは？ 140<CR>  
ヘッドは？ 3<CR>

インデックスからの距離(バイト)は？ 8008<CR>

ディフェクト 1: シリンダ 100 ヘッド 0 6535 バイト

ディフェクト 2: シリンダ 140 ヘッド 3 8008 バイト

これでよいですか？(Y/N) Y<CR>

ハードディスクの媒体検査を行いますか？(Y/N) Y<CR>

媒体検査を何回繰り返しますか？ 1<CR>

パーティションをいくつにしますか？(1~12) 2<CR>

残りセクタ数は、40326(20646912 バイト)です。

パーティション 1 に何セクタ割り当てますか？(最大 40326) 20163<CR>

残りセクタ数は、20163(10323456 バイト)です。

パーティション 2 に何セクタ割り当てますか？(最大 20163) 20163<CR>

パーティション 1: 20163 セクタ(10323456 バイト)

パーティション 2: 20163 セクタ(10323456 バイト)

残りセクタ: 0

これでよいですか？(Y/N) Y<CR>

これで、フォーマットが終了します。

## Easy Hard

SCSI バスへのインターフェースを行うためには、なにがなんでも SCSI コントローラ・チップを使わなければならないわけではありません。安易に汎用ロジックの 8 ビット・ラッチとバス・ドライバなどだけでも実現することができます。

そのような発想からパソコン通信のネット上で発表されたのが Easy Hard です。

もっとも Easy Hard は夕陽郎というハンドル・ネームの方のアイデアで、IBM-PC コンパチのプリンタ・ポートへ SCSI のハードディスクをケーブルだけで接続してしまうというものです。

なぜ、このようなことが可能であるかという、もともと IBM-PC のプリンタ・ポートは図 10 のようにデータが双方向の入出力が可能のように設計されているためで、これにプリンタの制御信号部分を SCSI の制御信号に割り当てることによって、SCSI のバス・フ

エーズをソフトによって制御することができるというわけです。

このため、ハードの製作は図 11 の変換ケーブルを 1 本作るだけで済んでしまいます。

図 12 にパラレル・インターフェース・レジスタを SCSI として使った場合の割り当てを示します(カッコ中は本来のプリンタ・ポートとしての割り当て)。

### ● ドライバ

対象とするシステムは現在のところ J3100SS/SL/GX/SGT101, IBM-PC/AT, 30Z-SX です。リスト 8~リスト 13 に全プログラム(ドライバ, フォーマット・プログラム)を示します。MAKEFILE を実行することで、.SYS と.EXE ができます。

8086 コードで書かれています。使用したアセンブラは MASM5.1 です。

対象になるドライブは ANSI の標準に準拠した SCSI ドライブ(CCS が使えるドライブ)です。

NEC PC9801-55 カードに接続できるドライブ、および富士通 FM-R/TOWNS シリーズに接続できるドライブなどであればまったく問題なく使用できます。

ただし、SCSI バスのパリティをディセーブルにしなければなりません。

具体的には、例えば D3841/D5852 などの NEC のドライブの場合にはたくさん並んでいるディップ・スイッチの中の PARITY と書いてあるスイッチを OFF にします。

富士通の M1055B ではディップ・スイッチの 5 番を OFF にします。M2246SA なら P と書いてあるジャンパを抜きます。

### ● ドライバの使用方法

ケーブルの製作が終わったら、まずプリンタ・ポートを双方向に設定します。

#### ・ J3100SS :

「SetupJ」でポートを「プリンタ・出力専用」に設定します。

#### ・ J3100SL :

「SetupJ」で「プリンタ接続：なし」を選択します。

#### ・ J3100GX :

プリンタ・コネクタ脇のディップ・スイッチ 2 番を ON にします。

#### ・ J3100SGT101 :

プリンタ・コネクタ脇のディップ・スイッチ 4 番を ON にします。

つぎに HDD を接続して SFEH.EXE を実行します。PC/AT などの漢字の出せない機械の場合には英語版 SFEHE.EXE を使用してください。

SFEH は MS-C で書いてあり、SCSI のハンドシェイクもすべて Cで行っているの、実行時に SCSI.

SYS がある必要はありません。

これが正常終了したら、ケーブルは多分正常なので、あとは CONFIG.SYS に SCSI.SYS を組み込めば終わりです。こちらは英語/日本語には関係ありません。

ターゲット ID, ロジカル・ユニット番号(LUN)は CONFIG.SYS の中でパラメータとして渡せます。

速い CPU のためにリセット後のディレイ時間調整を入っており、

例： DEVICE=SCSI.SYS /I4 /L2 /W4

ID4, LUN2 を選択、リセット後のウェイト 4 と指定します。

パラメータを付けない場合のデフォルトのターゲット ID は 0, ターゲットの LUN も 0 です。ID は 0 から 6 まで、LUN は 0 から 3 までを受け付けます。

リセット後のウェイト時間のデフォルトは 2 です。値は 1 から 9 までを受け付けます。

I と L と W 以外のパラメータは受け付ません。

なお、起動時に論理ブロック 0 を読み込んでパーテーション情報などをセットしているので、起動時には必ず HDD がレディ状態になっている必要があります。

### ▶ 注意

この接続の場合、電源 OFF 時に一瞬バス・フェーズが不安定になることがあるようで、ドライブによってはいったん回転を停止してしまいます。

その場合、リセットをかけたときに一度タイム・アウトになり、イニシャライズ・エラーが出る場合があります。これは避けようがないので、あきらめて再度リセットをしてください。

リセットをせずにファイル・アクセスなどをする場合にはエラー表示のときにリトライを選択すればそのまま続行できると思います。

J3100SL/SS の場合、レジャーームという厄介な機能があります。

電源切断中に HDD をつなぎ替えていない場合については対応していますが、違う HDD を接続した場合には必ず一度リセットを行ってください。場合によっては FAT を破壊することがあると思います(そんな恐ろしいこと自分ではとても試せない)。

このドライブは 256/512 セクタ両用になっています。起動時に勝手に判断するのでユーザはフォーマット時にどちらかを選んでください。

ターゲット ID というのは SCSI コントローラ(ドライブ)の SCSI バス上での ID です。7 はパソコンが使うので、0 から 6 までです。

LUN というのは、SCSI コントローラとハードディスク・ドライブ間での機番です。ST506 インターフェースの HDD の場合、ドライブの機番が 1 から 4 まで選べますが、これが 1 のときに LUN=0 です。SCSI 内蔵型 HDD の場合は必ず 0 になります。



このドライバ単体ではマルチ・ターゲット(複数台の HDD)はサポートしていませんが、SCSI ID か LUN を変更して CONFIG.SYS に登録すれば複数台接続も可能です。若干メモリの無駄になりますが、1 ドライブあたり 1.5K バイト位なので我慢できる範囲だと思います。

しかし、ケーブル長が延びることになるので、避けた方が無難です。MS-DOS はデータ化けについてはノーチェックです。

パーティションは 12 まで切れます。それぞれ最大 32M バイトまでです。ドライバ・イニシャライズのときにドライブの論理セクタ 0 (DD セクタ)を読み込んで、それから判断するのでブートの際に必ずデバイスがレディになっている必要があります。

くりかえしますが、前述のように MS-DOS には、「標準で組み込まれているデバイスよりも大きなセクタ・サイズのブロック・デバイスは追加できない」という闇の掟があります。

ですから、英語 DOS などでは 512 バイト/セクタ以上にできないので、32M バイト以上のパーティションが切れません(256 しか使えない OS9 よりはましかも?)。

もっとも日本語 DOS しか絶対に使用しない、というのなら、1024 バイト/セクタに見せることもできるので 64M バイトのパーティションも可能ですが……。

DD セクタの構成はつぎの通りです。(論理ブロック 0)

2 バイト: パーティションの個数

4 バイト \* 12: それぞれのパーティションの開始する論理ブロック・アドレス

14 バイト: 未使用

16 バイト \* 12: パーティションごとの B P B 情報  
合計 256 バイト

やはり、ソフト転送しかできないので非常に遅くなります。70K バイト/s 程度しか出ません。M1055B の場合、インターリーブは 8 が最適でした。

SPC の場合には J3100SS でも 300K バイト/s 近く出ますから、約 1/4 の実力といったところです。

簡単さを取るか、性能を取るかの問題で、J3100SS ならこの位でも十分かなあ?とは思っています(小さいファイルのアクセスならほとんど変わらないし)。

純正 PC/AT (286/8MHz) でもなぜか 100K バイト/s 位です。不思議だなあ?

### ● Easy Hard のフォーマット・プログラム

SPC ドライバとデータ互換があるので、こちらでフォーマットしたドライブは SPC カードでそのまま使えます。

以下に手順を示しますが、基本的な操作は SPC と

同じです。

M1055B+D3126 で、初めてフォーマットを行う場合の操作例を示します。ここでは

▶ SCSI ID, LUN は 0, ディフェクト個所が 2 箇所ある

▶ セクタ・サイズは 512 バイトに設定

▶ 媒体検査は 2 回行う

▶ 10M バイトずつふたつのパーティションに分割という操作を行っています。

<CR>が付いている部分が入力部分です。

フォーマットしたいドライブの SCSI-ID は何番ですか? 0<CR>

ロジカル・ユニット番号(ドライブの番号)は何番ですか? 0<CR>

物理フォーマットを行いますか? (Y/N) Y<CR>  
セクタ・サイズをいくつにしますか? (256・512)

512<CR>

インターリーブ・ファクタは、いくつにしますか?

8<CR>

一度もフォーマットを行っていないドライブですか? (Y/N) Y<CR>

このドライブの、シリンダ数はいくつですか?

615<CR>

ヘッド数はいくつですか?

4<CR>

書き込み位相補正(ライト・プリコンペ)を始めるシリンダはいくつですか?(不要な場合は、シリンダ数と同じ数) 256<CR>

書き込み電流補正(リデュースド・ライト・カレント)を始めるシリンダはいくつですか?(不要な場合は、シリンダ数と同じ数) 615<CR>

これでよいですか? (Y/N) Y<CR>

ディフェクトの個数はいくつですか? 2<CR>

ディフェクト 1: シリンダは? 100<CR>  
ヘッドは? 0<CR>

インデックスからの距離(バイト)は? 6535<CR>

ディフェクト 2: シリンダは? 140<CR>  
ヘッドは? 3<CR>

インデックスからの距離(バイト)は? 8008<CR>

ディフェクト 1: シリンダ 100 ヘッド 0  
6535 バイト

ディフェクト 2: シリンダ 140 ヘッド 3  
8008 バイト

これでよいですか? (Y/N) Y<CR>

ハードディスクの媒体検査を行いますか? (Y/N) Y<CR>

媒体検査を何回繰り返しますか? 2<CR>

パーティションをいくつにしますか? (1~12) 2<CR>  
残りセクタ数は、40326(20646912 バイト)です。

パーティション 1 に何セクタ割り当てますか?(最大

40326)

PAGE	66	132
MB89352 (SCS I プロトコルコントローラ) ドライバ		
copyright by 日立		
V1.0 July. 25, 1989 V1.1 July. 28, 1989 V1.2 Aug. 02, 1989		
V1.3 Aug. 03, 1989		
V1.4 Sep. 08, 1989		
V1.45 Nov. 07, 1989		
V1.5 Mar. 01, 1990		
V1.6 Jul. 28, 1990		
V1.7 Oct. 17, 1990		
抑型フォーマットは 256 * 512 * 1024 バイト / セクタ		
シングルターゲット (ターゲット ID、LUN は Config.sys で定義)		
ハードウェアはドライブあたり 12 マスで、パーティションサイズはおのり		
最大 3.2 (256/512 Bytes/Set時) * 6.4 (1024 Bytes/Set時) MB まで		
IFDEF 1186 .186		
ENDIF INCLUDE SFC.INC		
CODE SEGMENT PUBLIC ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE		
デバイスヘッダ		
DISKDEV LABEL WORD DW 2800H DOW STRATEGY DOW DSK_INT		
PAGE		
ワークエリア		
PTESAV DD 0 TMPHPA DB ?		
リクエストヘッダアドレス退避用		
SCSI COMMAND BUFFA (リード/ライト時)		
SCSI SCSIWORK < >		
BW FLG DB ? VFVF FLG DB ? UNIT NO DB ? SEC CNT DB ? TFRMODE DB ? SSIZ DB ? INIBIT DB ? INIGBIT DB ? LUN DB ?		
D D ステータスエリ		
ここに起動時に論理ブロック 0 を読み込んで来る		
ここは、パーティション情報等が入る (64 バイト)		
ここからパーティション毎の BPB が入る (16 * 12 バイト)		
DSPARAMS < > DUSCT		

```
40326)                                20163<CR>
残りセクタ数は、20163(10323456 バイト)です。
パーティション 2 に何セクタ割り当てますか？(最大
20163)                                20163<CR>
```

```
パーティション 1: 20163 セクタ (10323456 バイト)
パーティション 2: 20163 セクタ (10323456 バイト)
残りセクタ: 0
```

これでよいですか？ (Y/N) Y<CR>  
これで、フォーマットが終了します。

## ● 注意点など

速度的には、SPC などを使った専用ハードウェアより遅いのですが、ケーブル1本だけでハードディスクが接続できてしまうため、インターフェース・ボードなどを内蔵できない東芝のダイナブックなどのノート型パソコンなどでは、一部には必需品とさえいわれているようです。

注意として、電気レベル的には本来 SCSI バスを駆動することを考えていないプリンタ・ポートを使うのですからしかたがないのですが、**SCSI のデータ・パリティがないので接続するドライブはそのチェックを行わないように、設定できるドライブあるいは無視するドライブを選ぶ必要があります。**

また、パリティ・チェックを行っていないためケーブルを極力短くすることも大切です。

クリスト1> MAKEFILE (SPC用)

```

spec186.sys : spec170.asm spec.inc
masm /J186 spec170.asm,spec186.obj;
link spec186;
del spec186.obj
exebin spec186.spec186.sys
del spec186.exe

spec86.sys : spec170.asm spec.inc
masm /J86 spec170.asm,spec86.obj;
link spec86;
del spec86.obj
exebin spec86.spec86.sys
del spec86.exe

spec186no.sys : spec170.asm spec.inc
masm /J186 /NO spec170.asm,spec186no.obj;
link spec186no;
del spec186no.obj
exebin spec186no.spec186no.sys
del spec186no.exe

spec86no.sys : spec170.asm spec.inc
masm /J86 /NO spec170.asm,spec86no.obj;
link spec86no;
del spec86no.obj
exebin spec86no.spec86no.sys
del spec86no.exe

cscli.obj : cscli.c spec.h
cl -Zp -c cscli.c

sform.exe : sform.c cscli.obj spec.h
cl -Zp -F 4000 -F sform.exe sform.c cscli.obj

sformno.exe : sformno.c cscli.obj spec.h
cl -Zp -F 4000 -F sformno.exe sformno.c cscli.obj

```











[illegible]

:転送フェーズレジスタのセット			
	PSCTL	CS	ES
	PUSH	MOV	BX, CQADDR
	POP	LEA	DI, [BX], MESS
	CALL	CALL	:メッセージバイトを得る
	GETP	TEST	:バスフリーフェーズ
	TEST	JNE	:ビジーが0になるまで待つ
	JNE	HDCDBF	
	GETP	AL, 20H	
	AND	HDCDBF1	
	JNE		
	PUTP	AL, ESNOTRDY	:デバイスビジー
	MOV	HDCDBF	
	JMP		
	GETP	PSCTL	:ディスコネクト
	AND	AL, 7FH	
	PSCTL		:バスフリー割り込み禁止
	GETP	PSCTL	
	TEST	PSCTL	:割り込みソースのクリア
	JNE	PSCTL	
	GETP		
	MOV	DI, BUFADDR	:バッファアドレスの復帰
	MOV	ES, BUFADDR+2	
	CLC		:正常終了
	RET		
	MOV		:異常終了
	MOV	DI, BUFADDR	
	STC	ES, BUFADDR+2	
	RET		
	PAGE		
	MANTR10:		
	PSCTL	DI	
	GETP	PSCTL	
	TEST	AL, 80H	:REQが1になるまで待つ
	JE	MANTR10	
	GETP	PSCTL	
	TEST	AL, TSIO	:DATA INかOUTかチェック
	JE	MANOUT	
	PUTP1	PSCMD, SSSETACK	:DATA IN フェーズ
	MOV	AL, DX	
	TEST	AL, 80H	:REQが0になるまで待つ
	JNE	MAN11	
	GETP	PSTEMP	:データを貰う
	STOSB		
	PUTP1	PSCMD, SSRESETACK	
	MOV	AL, DX	
	TEST	AL, 88H	
	JE	MANTRFEXIT	:バスフリー?
			:の時はEXIT





＜リスト2＞ SPC170.ASM (つづき)

[illegible][illegible]



＜リスト2＞ SPC170.ASM (つづき)

	LEA	DI, DISCT	
	Mov CALL	TFMODE, 00H	: マニユアル転送を選択
	JC HDCMD		
	DIER		
	CMP SC3I_STAT, 00H		: ステータスチェック
	JE DEX1		: ノーエラー
			: センステータを得る
	CALL JC		
	Mov MOV	DIER AL, BYTE PTR SENS DAT+2	: リカバーデエラーは無視する
	CMP CMP	AL, 01H	
	JE JE	DIE1 AL, 06H	: リセットコンディションだともう一度
	CMP CMP	DINI1	
	JE JE		
DIER:	LEA	DX, MSGERR	: 失敗のメッセージを表示
	Mov MOV	AH, 09H	
	INT INT	21H	
	PUSH DS		
	Mov MOV	AL, 0	
	LDS LDS	BA, [PTR SAV]	: リクエストヘッダアドレスを得る
	Mov MOV	[BX, MEDIA], AL	: ドライブ数-0
	Mov MOV	WORD PTR [BX, TRANS], OFFSET DSK_INIT	
	Mov MOV	WORD PTR [BX, TRANS+2], CS	
	Mov MOV	WORD PTR [BX, COUNT], OFFSET INIT_TBL	
	Mov MOV	WORD PTR [BX, COUNT+2], CS	: B P B 配列のポインタ
	POP POP	DS	
	STC		
	Mov MOV	AL, ESNOTRDY	: エラーコードのセット
	JMP JMP	ERR_EXIT	
DEX1:	LEA	DX, MSGCMP	: 成功のメッセージを表示
	Mov MOV	AH, 09H	
	INT INT	21H	
	Mov MOV	CH, 0	
	Mov MOV	CL, BYTE PTR DISCT_DRIVEIN	: D D セクタから、パーティションの数を得る
	Mov MOV	DL, BYTE PTR STDRV	
	ADD ADD	DL, A	
	LEA LEA	BX, DSXBPB0+9	
DEX2:	PUSH DX		: ドライブ番号の表示
	Mov MOV	AH, 02H	
	INT INT	21H	
	Mov MOV	DL, :	
	INT INT	21H	
	Mov MOV	AL, BYTE PTR [BX]	
	Mov MOV	AH, BYTE PTR SSIZ	
	CMP CMP	AH, 04H	: 1024bytes/sectorの場合
	JE JE	DIE3	
	SHR SHR	AL, 1	
DEX3:	Mov MOV	AH, 0	
	ADD ADD	AH, 02H	: 1MB以下を四捨五入
	SIR SIR	AX, 1	
	AND AND	AX, 0FFh	: AX = Drive_Capacity(MB) * 2
	PUSH BX		
	LEA LEA	BX, CAPTABLE	
	ADD ADD	DX, AX	
	Mov MOV	WORD PTR [BX]	
	Mov MOV	AH, 02H	
	INT INT	21H	
	Mov MOV	DL, BYTE PTR [BX+1]	
	INT INT	21H	
	POP POP	BX	
	LEA LEA	DX, MSGCAP	
	INT INT	21, 09H	
	ADD ADD	EAX, 1	: B P B アドレスを進める
	ADD ADD	BX, 16	: ドライブ番号 + = 1
	INC INC	DL	

[illegible]

<リスト3> SPC. INC

SPCHAS	EQU	0300H	: SPCのベースアドレス
ISINIBIT	EQU	7	: イニシエータID
ISINIBIT	EQU	80H	
ISAGBIT	EQU	00	: ターゲットID (デフォルト値0)
ISAGBIT	EQU	01H	
SJLN	EQU	00H	: ロジカルユニット番号 (デフォルト値0)

## SPICのI/Oアドレス定義

## SPCのコマンド定義

## SCSIのコマンド定義

リクエストヘッダ構造体の定義

## SCSIコマンドバケット構造体の定義

SCS WORK	STRUC	?	?	?	?	?	?	?	?
CMDS	DB								
SECH	DB								
SECH	DB								
SECH	DB								
SECL	DB								
ELOCK	DB								



<リスト4> SCSI.C

```

/* Format program for MB9352( SPC ) on J-3100
   Converted from 63C09E Assembler ( Written by Angie ) by MAL5
   SCS1 handling routines

```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <_spec.h>
```

```
long l, t;
printf( "VnCSS | バスリセットを実行中です。10秒お待ち下さい。Vn" );
```

EQU	00H	:WRITE PROTECT
EQU	01H	:ILLEGAL UNIT
EQU	02H	:NOT READY
EQU	03H	:ILLEGAL UNIT
EQU	04H	:CRC ERROR
EQU	05H	:ERROR & REQ. LENGTH
EQU	06H	:SEEK ERROR
EQU	07H	:MEDIA ERROR
EQU	08H	:SECTOR NOT FOUND
EQU	09H	:PAPER OUT
EQU	0AH	:WRITE ERROR
EQU	0BH	:READ ERROR
EQU	0CH	:GLOBAL ERROR
EQU	0FH	:MEDIA CHANGE ERROR

1/Oポートアクセスのマクロ定義

CLRP	MACRO	ADDR	ADDR	MACRO	ADDR
	MOV	DX, ADDR		MOV	DX, ADDR
	IN	AL, DX		IN	DX, AL
	MOV	AL, 0		MOV	AL, AL
	OUT	DX, AL		OUT	DX, AL
	ENDM			ENDM	
GETP	MACRO	ADDR	ADDR	MACRO	ADDR
	MOV	DX, ADDR		MOV	DX, ADDR
	IN	AL, DX		IN	DX, AL
	ENDM			ENDM	
PUPP	MACRO	ADDR	ADDR	MACRO	ADDR
	OUT	DX, ADDR		OUT	DX, AL
	OUT	DX, AL		OUT	AL, CL
	ENDM			ENDM	CL
PUPPI	MACRO	ADDR, DATA		MACRO	
	MOV	AL, DATA		MOV	
	OUT	DX, ADDR		OUT	
	OUT	DX, AL		OUT	
	ENDM			ENDM	
PUPPW	MACRO	ADDR	ADDR	MACRO	ADDR
	MOV	DX, ADDR		MOV	DX, ADDR
	OUT	DX, AL		OUT	DX, AL
	MOV	AL, AL		MOV	AL, AL
	OUT	DX, AL		OUT	AL, CL
	ENDM			ENDM	CL

```
static struct {
    int
    char
    ssiz:
    altsiz:
```





```

/* Test Unit ready */
testready( id, lun )
{
    int id, lun;

    unsigned char v[ 6 ], buf[ 80 ];
    int i;

    v[ 0 ] = C.TSTRDY;
    v[ 1 ] = ( lun << 5 );
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;

    dma = 0; /* not DMA mode */

    return ( hcmd( id, v, buf ) );
}

/* read capacity */
readcapacity( id, lun, buf )
{
    int id, lun;
    unsigned char *buf;

    unsigned char v[ 10 ], sensbuf[ 256 ];

    v[ 0 ] = C.RDCAP;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00; /* logical block address is 0 */
    v[ 6 ] = v[ 7 ] = 0x00; /* PHL bit false */
    v[ 8 ] = v[ 9 ] = 0x00; /* flag and Thk false */

    dma = 0; /* not DMA mode */
    if( hcmd( id, v, buf ) ){
        if( request_sense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hcmd( id, v, buf );
        }
        else{
            printsens( sensbuf );
        }
    }
}

/* Write 1 Sector */
write1( id, lun, lsn, buf )
{
    int id, lun;
    lsn;
    unsigned char *buf; /* data buffer */

    unsigned char v[ 6 ];
    unsigned char sensbuf[ 256 ];

    if( scsiz == 256 ){
        lsn = 2L;
    }
    v[ 0 ] = C.WRTSEC;
    v[ 4 ] = ( ( scsiz == 256 ) ? 2 : 1 );
    v[ 5 ] = 0x00;
    v[ 1 ] = ( lun << 5 ) + ( lsn >> 16 ) & 0x00ff;
    v[ 2 ] = ( lsn >> 8 ) & 0x00ff;
    v[ 3 ] = lsn & 0x00ff;
    dma = 0;
    if( hcmd( id, v, buf ) ){
        if( request_sense( id, lun, sensbuf ) == 6 ){
            /* reset condition */
            hcmd( id, v, buf );
        }
        else{
            printsens( sensbuf );
        }
    }
}

request_sense( id, lun, buf )
{
    int id, lun;
    unsigned char *buf;

```

```

/* check error, do request sense */
unsigned char v[ 6 ];

v[ 0 ] = C.RESEN;
v[ 1 ] = lun << 5; /* set Lun */
v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
v[ 4 ] = 255;
dma = 0x00;
hcmd( id, v, buf );
return( buf[ 2 ] );
}

printsens( buf )
{
    unsigned char *buf;

    printf( "Sense key = %02X, Extend sense code = %02X\n", buf[ 2 ], buf[ 12 ] );
    dump( buf );
    exit( 99 );
}

/* output command to HDD */
hcmd( id, sc, buf )
{
    int id;
    unsigned char *sc, *buf;

    int treg; /* temp. reg. */
    int idb;

    idb = 0x0001 << id;

    /* selection phase */
    outp( scsibase + S_PCTL, 0x00 );
    treg = inp( scsibase + S_BID );
    outp( scsibase + S_BID, treg ); /* output target & init. ID */
    outp( scsibase + S_TCH, 0x0f ); /* Set timeout Counter */
    outp( scsibase + S_TCK, 0x46 );
    outp( scsibase + S_TCL, 0x04 );
    outp( scsibase + S_INTS, 0x00 );
    outp( scsibase + S_SCMD, SELECT );
    while( !( treg = inp( scsibase + S_INTS ) ) & 0x14 ){
        /* wait IRQ */
    }

    if( treg != 0x10 ){
        printf( "Device Busy at selection phase.\n" );
        exit( 99 );
    }

    /* command phase */
    do{
        while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
            /* wait until REQ = 1 */
        }
        treg &= ( MSG + CD + IO );
        while( treg != CD );
        outp( scsibase + S_PCTL, treg | 0x80 ); /* bus free IRQ disable */
        mantfr( sc );

        /* data transfer phase */
        while( !( treg = inp( scsibase + S_PSNS ) ) & 0x80 ){
            /* wait until REQ = 1 */
        }
        if( !( treg & ( MSG | CD ) ) ){ /* status phase? */
            /* no, data phase. */
            treg &= ( MSG | CD | IO );
            treg != 0x80; /* bus free IRQ enable */
            outp( scsibase + S_PCTL, treg );
            mantfr( buf );
        }

        /* status phase */
        do{
            while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ){
                /* wait until REQ = 1 */
            }
            treg &= ( MSG | CD | IO );
            while( treg != ( CD | IO ) );
        }
    }
}

```

```

treg |= 0x80;
outp( scsibase + S_PCTL, treg ); /* bus free IRQ enable */
tfrsub( &tbuf ); /* get Status byte */
/* message phase */
do
    while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ) {
        /* wait until REQ = 1 */
    }
    treg &= ( MSG | CD | 10 );
    while( treg != ( MSG | CD | 10 ) );
    treg |= 0x80;
    outp( scsibase + S_PCTL, treg ); /* bus free IRQ enable */
    tfrsub( &asin ); /* get message byte */
/* bus free phase */
while( inp( scsibase + S_PSNS ) & 0x08 ) {
    /* wait until BUSY = 0 */
}
if( !( treg = ( inp( scsibase + S_INTS ) & 0x20 ) ) ) {
    outp( scsibase + S_INTS, treg );
    printf( "Device Busy at bus free phase. Yn" );
    exit( 99 );
}
/* disconnect */
treg = inp( scsibase + S_PCTL ) & 0x7f;
outp( scsibase + S_PCTL, treg ); /* bus free interrupt disable */
treg = inp( scsibase + S_INTS );
outp( scsibase + S_INTS, treg ); /* reset interrupt source */
treg = inp( scsibase + S_SCTL );
if( treg & 0x01 ) {
    /* parity error */
    printf( "parity error occurred. Yn" );
    exit( 99 );
}
else if( !( treg & 0x02 ) ) {
    /* no error */
    return( 0 );
}
else {
    return( 1 );
}
}

/* manual Data transfer routine */
mainfr( buf )
unsigned char *buf;
{
    int treg; /* temporary register */
    do {
        tfrsub( buf++ );
        while( !( ( treg = inp( scsibase + S_PSNS ) ) & 0x80 ) ) {
            /* wait until REQ = 1 */
        }
        treg &= ( MSG | CD | 10 );
        while( treg == ( inp( scsibase + S_PCTL ) & ( MSG | CD | 10 ) ) );
        return( 0 );
    }
    /* sub routine: 1 byte transfer */
    tfrsub( buf );
    char *buf;
    int treg; /* temp. reg. */
    while( !( ( inp( scsibase + S_PSNS ) & 0x80 ) ) ) {
        /* wait until REQ = 1 */
    }
    if( inp( scsibase + S_PCTL ) & 10 ) {

```

```

/* I/O IN */
outp( scsibase + S_SCMD, SETACK );
while( inp( scsibase + S_PSNS ) & 0x80 ) {
    /* wait until REQ = 0 */
}
tbuf = inp( scsibase + S_TEMP );
}
else {
    /* I/O OUT */
    outp( scsibase + S_TEMP, tbuf );
    outp( scsibase + S_SCMD, SETACK );
    while( inp( scsibase + S_PSNS ) & 0x80 ) {
        /* wait until REQ = 0 */
    }
    outp( scsibase + S_SCMD, RESETACK );
}
/* SPC initialize */
init() {
    int i;
    outp( scsibase + S_BDD, INITID ); /* controller ID = 7 */
    outp( scsibase + S_SCMD, 0x00 );
    outp( scsibase + S_TMDD, 0x00 );
    outp( scsibase + S_PCTL, 0x00 );
    outp( scsibase + S_PCTL, 0x00 );
    outp( scsibase + S_TEMP, 0x00 );
    outp( scsibase + S_TCH, 0x00 );
    outp( scsibase + S_TCH, 0x00 );
    outp( scsibase + S_PSNS, 0x00 );
    outp( scsibase + S_SCTL, 0x88 );
    for( i = 0; i < 16; i++ ) {
        /* for delay */
    }
    outp( scsibase + S_SCTL, 0x18 );
}
dump( buf )
unsigned char *buf;
{
    int i, j;
    unsigned d;
    printf( " 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEFYn" );
    for( i = 0; i < 256; i += 16 ) {
        printf( "%04X:", i );
        for( j = 0; j < 16; j++ ) {
            printf( "%02X", buf[ i + j ] );
        }
        for( j = 0; j < 16; j++ ) {
            d = buf[ i + j ];
            if( ( d >= 0x20 ) && ( d < 0x7f ) ) {
                putchar( d );
            }
            else {
                putchar( '.' );
            }
        }
        putchar( '\n' );
    }
}

```



リスト5> SPC.H

```

/* Defs */
/* SPC Register */
#define S_B0ID 0x00
#define S_SCTL 0x01
#define S_SQMD 0x02
#define S_QMD 0x03
#define S_QMIS 0x04
#define S_SQCS 0x05
#define S_SQCS 0x06
#define S_SSTS 0x07
#define S_SCTL 0x08
#define S_WIC 0x09
#define S_DRG 0x0a
#define S_DRP 0x0b
#define S_TCM 0x0c
#define S_TCM 0x0d
#define S_TCL 0x0e

/* SPC Command */
#define SELECT 0x20
#define RESET 0x00
#define SETTAG 0x00
#define RESET 0x10

/* transfer Mode */
#define MS 0x04
#define CD 0x02
#define IO 0x01

/* Controller command */
#define C_TRDP 0x00
#define C_ZERO 0x01
#define C_RESET 0x03
#define C_ERROR 0x04
#define C_WTSEC 0x04
#define C_WTSEC 0x04
#define C_WTSEC 0x08
#define C_WTSEC 0x15
#define C_WTSEC 0x1a
#define C_WTSEC 0x25

/* SCSI ID */
#define INITID 7
#define INITBIT 0x80
#define TAGTID 0
#define TAGTBIT 0x01

/* use DMAC Ch.1 */
#define D_ADDR 0x0002 /* c.l address register */
#define D_COUNT 0x0003 /* c.l count register */
#define D_STATS 0x0008
#define D_CMD 0x0008
#define D_RST 0x0009
#define D_MASK 0x000a
#define D_CLR 0x000c
#define D_MASK 0x000c
#define D_BANK 0x0003

/* Global Variables */
/* 6 bytes length command table */
typedef struct {
    unsigned char cmd; /* 0n Code */
    unsigned char scch; /* LSN High */
    unsigned char sccl; /* LSN Mid */
    unsigned char block; /* LSN Low */
    unsigned char ctrl; /* control byte */
} SCSI_CMD;

/* data buffer for system */
char sysbuf[2048];

char stbuf; /* buffer for status byte */
char asgin; /* buffer for message byte */
char ints; /* interrupt status */
int daa;
int daamode;
int scsmode;

```

リスト6> SFORM.C

```

/* Format program for M89332(SPC) on J-3100
Converted from 63C09E Assembler ( Written by Angie ) by MALS
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "spc.h"

unsigned scsibase;
int scsisz;

main()
{
    unsigned char buf[512];
    int i, id, lun, int, patno;
    long l, maxset, att;

    printf(" SPCカード用 HDDイニシャライズユーティリティ V1.6 Vn\n");
    printf(" Copyright by Tsuru-20 Jul.28,1990Vn\n");
    printf(" ** 注意 ** 実行すると、今HDDに入っているデータは全て破壊されます。 Vn\n");
    printf(" ** 途中で中断したい時は、Ctrl-Cを押し下さい。 Vn\n");

    printf(" SPCカードの I/O アドレスはどこになっていますか? - ");
    scanf("%x", &scsibase);

    printf(" Vn フォーマットしたいドライブの SCSI I - ID は何番ですか? - ");
    scanf("%d", &id);
    printf(" Vn ロジカルユニット番号 (ドライブの番号) は何番ですか? - ");
    scanf("%d", &lun);

    init();
    reset(id, lun);

    printf(" セクタサイズ、インタリーブ値を変更する場合は物理フォーマットを行って下さい。 Vn\n");
    printf(" Vn 物理フォーマットを行いますか? (Y/N) - ");
    scanf("%s", buf);
    if( #buf == 'y' || #buf == 'Y' ) {
        physformat(id, lun);
    }

    readcapacity(id, lun, buf);
    if( #buf[6] != 1 ) && ( #buf[6] != 2 ) && ( #buf[6] != 4 ) {
        printf(" Vn ** 物理セクタサイズが256, 512, 1024バイト以外になっています。 Vn\n");
        exit(1);
    }
    if( scsisz == 0 ) {
        if( scsisz == buf[6] * 256 ) {
            if( #buf[6] * 256 != scsisz ) {
                printf(" Vn ** 選択されたセクタサイズと物理セクタサイズが異なっています。 Vn\n");
                printf(" ** 選択されたセクタサイズ: %d 物理セクタサイズ: %d Vn\n");
                scsisz = buf[6] * 256;
            }
            scsisz = buf[6] * 256;
            printf(" ** 選択して処理を行いますか? (Y/N) - ");
            scanf("%s", buf);
            if( #buf == 'y' ) && ( #buf != 'Y' ) {
                exit(1);
            }
        }
        maxset = 65536L * #buf[1] + 256L * #buf[2] + #buf[3] * 1L;
        printf(" Vn Vn ハードディスクの媒体検査を行いますか? (Y/N) - ");
        scanf("%s", buf);
        if( #buf == 'y' ) {
            #buf = 'Y';
            verifyunit(id, lun, maxset);
            makepart(id, lun, maxset);
        }
        printf(" Vn フォーマットが完了しました。 Vn\n");
        printf(" Config.sysに SPC6186.SYS を登録し、リセットして下さい。 Vn\n");
    }
}

```





# リスト6> SFORMM.C (つづき)

```

buf[ 18 ] = 0x00; /* write precomp start cylinder */
buf[ 19 ] = wp >> 8;
buf[ 20 ] = wp & 0x00ff;
buf[ 21 ] = 0x00; /* reduced write current start cylinder */
buf[ 22 ] = rwc >> 8;
buf[ 23 ] = rwc & 0x00ff;

/* page code 3 */
buf[ 24 ] = 0x03;
buf[ 25 ] = 0x16;
buf[ 26 ] = 0x00;
buf[ 27 ] = 0x00;
buf[ 28 ] = 0x00;
buf[ 29 ] = 0x00;
buf[ 30 ] = 0x00;
buf[ 31 ] = 0x00;
buf[ 32 ] = 0x00;
buf[ 33 ] = 0x00;
buf[ 34 ] = 0x00;
buf[ 35 ] = 0x00;
buf[ 36 ] = 0x00;
buf[ 37 ] = 0x00;
buf[ 38 ] = 0x00;
buf[ 39 ] = 0x00;
buf[ 40 ] = 0x00;
buf[ 41 ] = 0x00;
buf[ 42 ] = 0x00;
buf[ 43 ] = 0x00;
buf[ 44 ] = 0x00;
buf[ 45 ] = 0x00;
buf[ 46 ] = 0x00;
buf[ 47 ] = 0x00;

/* page code 1 */
buf[ 48 ] = 0x01;
buf[ 49 ] = 0x08;
buf[ 50 ] = 0x24;
buf[ 51 ] = 0x08;
buf[ 52 ] = 0x00;
buf[ 53 ] = 0x00;
buf[ 54 ] = 0x00;
buf[ 55 ] = 0x00;

if( hdcmf( id, v, buf )){
    if( requestSense( id, lun, sensbuf ) == 6 ){
        /* reset condition */
        hdcmf( id, v, buf );
    }else{
        printSense( sensbuf );
    }
}

/* Format Unit */
formatfirst( id, lun, invl );
int id, lun, invl;

unsigned char v[ 6 ], sensbuf[ 256 ];
struct{
    int dfno, i;
    int cyl;
    int bkt;
    int bld;
    int diff;
}defect[ 100 ];
char buf[ 1024 ], *ptr;

printf( "インデックスの開始はいくつですか? " );
scanf( "%d", &dfno );
for( i = 0; i < dfno; i++){
    printf( "インデックス %d: シリンダは? ", i + 1 );
    scanf( "%d", &(defect[ i ].cyl) );
    printf( "ヘッドは? " );
    scanf( "%d", &(defect[ i ].bkt) );
    printf( "インデックスからの距離(バイト)は? " );
    scanf( "%d", &(defect[ i ].diff) );
}
putchar( '\n' );
for( i = 0; i < dfno; i++){

```

```

printf( "インデックス %d: シリンダ %d ヘッド %d バイト %d\n",
    i + 1, defect[ i ].cyl, defect[ i ].bkt, defect[ i ].diff );
}
printf( "\nこれで良いですか? (Y/N) " );
scanf( "%s", buf );
while( *buf != '\0' && *buf != 'Y' );

v[ 0 ] = C.FMTDRV;
v[ 1 ] = ( (lun < 5) + 0x1e;
v[ 2 ] = v[ lun < 5 ] + 0x00;
v[ 4 ] = invl;

buf[ 0 ] = 0x00; /* create p-list */
buf[ 1 ] = 0xf1; /* # 8 >> 8;
buf[ 2 ] = { dfno * 8 } & 0x00ff;
buf[ 3 ] = { dfno * 8 } & 0x00ff;

ptr = &( buf[ 4 ] );
for( i = 0; i < dfno; i++){
    *ptr++ = defect[ i ].cyl >> 8;
    *ptr++ = defect[ i ].cyl & 0x00ff;
    *ptr++ = defect[ i ].bkt;
    *ptr++ = defect[ i ].diff >> 8;
    *ptr++ = defect[ i ].diff & 0x00ff;
    *ptr++ = 0;
    *ptr++ = 0;
    *ptr++ = 0;
}

dma = 0; /* not DMA mode */
printf( "インデックスフォーマットを行っています。2～5分程度お待ち下さい。 \n" );
if( hdcmf( id, v, buf )){
    if( requestSense( id, lun, sensbuf ) == 6 ){
        /* reset condition */
        hdcmf( id, v, buf );
    }else{
        printSense( sensbuf );
    }
}

/* Format program for MB89352( SPC ) on J-3100
Converted from 6309E Assembler ( Written by Angle ) by MAL5
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <spc.h>

unsigned sensbase;
int setsiz;

main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, invl, patno;
    long l, basect, stl;

    printf( "SPCカード用 MO-Diskインデックスユーティリティ V.1.6 \n" );
    printf( "Copyright by Tsun-20 Jul. 28, 1990 \n" );
    printf( " ** 注意 ** 実行すると、今MOに入っているデータは全て破壊されます。 \n \n" );
    printf( "途中で中断したい時は、Ctrl-Cを押して下さい。 \n \n" );
    printf( "SPCカードの I/O アドレスはどこになっていますか? " );
    scanf( "%x", &sensbase );
}

```

## リスト7> SFORMMO.C

```

/* Format program for MB89352( SPC ) on J-3100
Converted from 6309E Assembler ( Written by Angle ) by MAL5
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <spc.h>

unsigned sensbase;
int setsiz;

main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, invl, patno;
    long l, basect, stl;

    printf( "SPCカード用 MO-Diskインデックスユーティリティ V.1.6 \n" );
    printf( "Copyright by Tsun-20 Jul. 28, 1990 \n" );
    printf( " ** 注意 ** 実行すると、今MOに入っているデータは全て破壊されます。 \n \n" );
    printf( "途中で中断したい時は、Ctrl-Cを押して下さい。 \n \n" );
    printf( "SPCカードの I/O アドレスはどこになっていますか? " );
    scanf( "%x", &sensbase );
}

```

クリスト7> SFORMMO.C (つづき)

```

print("Yn フォーマットにたいドライブのSCSI-IDは何番ですか？ ");
scanf("%d",&id);
scanf("%d",&id);
scanf("%d",&id);
scanf("%d",&id);

init0();

reset(id, lun);

print("Yn 新品の媒体の場合、メーカーがあらかじめフォーマットされて'vn'
print('いない場合もありますので、一度物理フォーマットを行なって'vn");
print('ください。また、24分程度かかります。'vn");
print("Yn 物理フォーマットを行いまさか？ (Y/N)");

scanf("%s",&buf);
if( buf == 'Y' )
{
    physformato(id, lun);
}

readcapacity(id, lun, buf);
setsz = buf[6] * 256;
maxsz = 65536; # buf[1] * 256; # buf[2] * 256; # buf[3] * 256;

print("YnYnYnMMOの媒体検査を行いますか？ (Y/N)");

scanf("%s",&buf);
if( buf == 'Y' )
{
    verifyunit(id, lun, maxsz);
}

makepart(id, lun, maxsz);

print("Yn フォーマットが完了しました。'vn");
print("Conf.sysにSP0.86mo/1.86mo.SYSを登録し、'vn");
print("一度リセットをして下さい。'vn");
}

/*Format Unit */
physformato(id, lun, initv)
int id, lun, initv;
{
    unsigned char v[ 6 ]; sensbuf[ 256 ];

    v[ 0 ] = C.FRWDR;
    v[ 1 ] = lun << 5;
    v[ 2 ] = lun << 5;
    v[ 3 ] = 0x03;
    v[ 4 ] = 0x05;
    v[ 5 ] = 0x00;

    dma = 0;
    /* not DMA mode */
    print("Yn物理フォーマットを行って下さい。24分程度お待ち下さい。'vn");
    if( !hdwtest(id, lun, sensbuf) )
    {
        if( request_sense(id, lun, sensbuf) == 6 )
        {
            /* reset condition */
            hdwmd(id, lun, sensbuf);
        }
        else
        {
            print(sensbuf);
        }
    }
}

```

## ＜リスト8＞ MAKEFILE (Easy Hard用)

```
scsi.sys : scsi.asm easyhard, inc
link scsi.asm:
  link scsi:
  exehin scsi: scsi.sys
  del scsi.obj
  del scsi.exe

sfch.exe : sfch.c easyhard, h
  cl -Zp -F 4000 sfch.c
  del sfch.obj

sfche.exe : sfche.c easyhard, h
  cl -Zp -F 4000 sfche.c
  del sfche.obj
```

<リスト9> SCSI.ASM

EASYPHARDドライバ for SCSI-CCS	copyright © 鶴城
V0.5 Feb. 11.1990	256/512画用化
V0.6 Feb. 19.1990	起動パターンをセクタ単位に一般リレーズ
V0.7 Feb. 22.1990	セクタ単位に一般リレーズに対応し、thanks for your mail
V1.1 Apr. 02.1990	セクタ単位の抽出、インデックス(Copy)に変更
V1.2 May. 05.1990	J 3100 SSI / Cx / SGTでも動作するように変更
V1.3 Mar. 07.1990	
物理フォーマットは2.56 * 5.12 バイト / セクタ	
シングルターゲッド (ターゲットID, LUNはSCSI, INCで定義)	
パーティションはドライブあたり12まで、パーティションサイズはおのり	
最大3.2MBまで	
INCLUDE EASYHARD.INC	
CODE SEGMENT PUBLIC ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE	
デバイスヘッダ	
DISKDEV LABEL WORD	-1 ; 最終デバイス、なししておく
DW DD	: プロテクトブル・システム・トリプルID
DW DD	: インタラプトル・タインメントリ
PAGE PAGE	
ワークエリア	
FITSAV DD ?	: リクエストヘッドアドレス追適用
TAPPIA DD ?	: Phase change の検出用
RSTWT DW 512	: リセット送時のウェイト時間
SCSI コメントバッファ (リード/ライト時)	
CSCLI SCSEWORK < >	
RW FLG DB ?	: 0の時リード、1の時ライト
VFV FLG DB ?	: 1の時ベリアライオン (未使用)
UNIT NO DB ?	: ユニコード番号
SEC CNT DW ?	: セクタカウンタ
TERMODE DB ?	: 転送モード (0: フォーマットする時、1: デバイスモード)
SIZ DB ?	: セクタあたりのバイト数 (256, 512の時が入る)
BIT BIT DB ?	: ターゲットIDビット
TAGTBIT DB ?	: ロジカルユニットナンプ
LUN DB DB	
DDセクタエリア	
ここに起動時に論理ブロック0を読み込んで来る	
ここは、パーティション情報等が入る (64*バイト)	
DOPPARAMS < >	
ここからパーティション毎のBPBが入る (16 * 12 バイト)	
DISCT	
DISKEPB0 BPBTYP <	: パーティション0
DISKEPB1 BPBTYP <	: パーティション1
DISKEPB2 BPBTYP <	: パーティション2
DISKEPB3 BPBTYP <	: パーティション3
DISKEPB4 BPBTYP <	: パーティション4
DISKEPB5 BPBTYP <	: パーティション5
DISKEPB6 BPBTYP <	: パーティション6
DISKEPB7 BPBTYP <	: パーティション7
DISKEPB8 BPBTYP <	: パーティション8
DISKEPB9 BPBTYP <	: パーティション9
DISKEPB10 BPBTYP <	: パーティション10
DISKEPB11 BPBTYP <	: パーティション11

DDセクタエリア  
ここに起動時に論理ブロック0を読み込んで来る  
ここは、パーティション情報等が入る (54バイト)

ここからパーティション毎のBRPBが入る(16\*12バイト)



[illegible][illegible]

＜リスト9＞ SCSI.ASM (つづき)

```

MOV SI, BX
JRS [BX, PC+1] AH
MOV SI, [BX, PC+1] SI
MOV [BX, COUNT+2], CS
JMP EXIT

; * コマンド4: インポート *****
DSX_READ:
MOV RW, 0
JMP RW_COMMON

; * コマンド8: アウトポート *****
DSX_WRT:
MOV RW, 1
JMP RW_COMMON

; * コマンド9: アウトポート & ペリファイ *****
DSX_WRT:
MOV RW, 1
JMP RW_COMMON

; * リード/ライトの共通ルーティン
RW_COMMON:
JCKZ EXIT
MOV UNIT, NO_AL
SEC CNT, CX

CMP RW, 0
JNE WRT

MOV SCS1, CMDS, CSISELSEC
JMP RW10

MOV SCS1, CMDS, CSWRTSEC

MOV AH, 0
SHL AX, 1
RAX, DSISCT, SCTOFS
ADD BX, AX

ADD DX, WORD PTR [BX]
MOV MOV, 0
ADC AL, BYTE PTR [BX+2]

MOV CMP, 0
RCL RDI, 0
SHL SI, 1
SHL DI, 1
RCL AL, 1

OR AL, BYTE PTR LUN
MOV SCS1, SECH, AL
MOV SCS1, SECH, DI
MOV SCS1, SECH, CL
MOV SCS1, CNTL, 0
LEA AX, SCS1

```

	MOV CALL	HIDCMD	TFRMODE, OFFH		:転送モードはアドレスに転送 : SCS I ハンドルルーティンを呼ぶ
	JC		RWER		: セレクションタイムアウト
	MOV CMP		AL, SCSI_STAT		: ノーエラーならそのままリターン
	JE		AL, 0		
			RWRTN		
		REGENS			: チェックの場合はリクエストセトスをして
	CALL JC		RWER		
	MOV AL		AL, SENS DAT+2		: リカバードエラーは無視する
	CMP CMP		AL, 1		
	JE JE		RWRTN		: リセットコンディションの時はリードライト
	CMP JE		AL, 6		: 出来てないからリトライ
			RWERTRY		
	CLC				
	JMP JMP		RWER		: それ以外は本当のエラー
					: リトライする
RWRTV:	LEA MOV		AX, SCSI		
	MOV CALL		TFRMODE, OFFH		
		HIDCMD			
	JC		RWER		
	MOV AL		AL, SCSI_STAT		
	CMP CMP		AL, 0		
	JNE JNE		RWER		: リトライはしない
					: 正常終了
RWRTN:	LDS MOV		BX, [PTRSAV]		: リクエストヘッダアドレスを得る
	MOV MOV		[BX, COUNT], CX		: 転送されたセクタ数をセット
	JMP JMP		EXIT		
RWER:	JNC		RWEI		: 異常終了
	MOV MOV		AL, ESNOTRDY		: A L にエラーコードをセットして
	JMP JMP		RWERET		
RWEI:	TEST RW		FLAG, 01H		
	JNE JNE		RWFZ		
	JMP JMP		AL, ESREAD		
			RWERET		
RWFZ:	MOV MOV		AL, ESWRITE		
RWERET:	RWERET		ERR_EXIT		: ERR_EXIT にジャンプ
	RET RET				
:					
:					
:					
:					
REGENS:	PUSH DI				
	PUSH ES				
	MOV MOV		REQS, CMD5, CSERSEN		: 専用バッファアドレスにコマンド
	MOV MOV		AL, BYTE PTR JUN		
	MOV MOV		REQS, SECH, AL		: バケツをセット
	MOV MOV		REQS, SECH, 00H		: この後でコマンドリットライをする
	MOV MOV		REQS, SECH, 00H		: 事があるのでバケツを分けてある
	MOV MOV		REQS, BLOCK, 128		
	MOV MOV		REQS, CNTL, 00H		
	LEA LEA		AX, REGS		
	LEA LEA		DI, SENS DAT		: ここにセンスデータが格納される
		CS			: CS -> ES
	PUSH POP		ES		
	MOV CALL		TFRMODE, 0		: ソフト転送モードを選択
	POP POP				
	POP POP				



＜リスト9＞ SCSI.ASM (つづき)

```

PAGE      RET
:
: SCS1ハンドリングルーチン
: AXにコマンドバケットアドレス
: ES: DIに転送バufferアドレス
:
HDCMD0:
MOV        CMDADR, AX
MOV        BUFADR, DI
MOV        BUFADR+2, ES
POP        CS
ES

GETP       PCNT
TEST       AL, TSREQ
CALL       SCS1_INIT
HDCMD0:
MOV        CX, 1000

GETP       PCNT
TEST       AL, TSREQ
JNE        JE
POP        JE
JMP        JMP

HDCMD2:
MOV        AL, BYTE PTR INITBIT
OR         AL, BYTE PTR TAGTBIT
NOT        AL
MOV        DX, PSDATA
OUT        DX, AL
POP        OUT
PUTP1      PCNT, TSREQ+TSSEL
HDCMD2:
MOV        CX, 4000

GETP       PCNT
TEST       AL, TSREQ
JNE        JE
POP        JE
JMP        JMP

HDCMD11:
GETP       PCNT
TEST       AL, TSREQ
JNE        JE
POP        JE
JMP        JMP

HDCMD10:
MOV        AL, ESNOTRDY
PUSH       AX
PUSH       CX
PUTP1      PCNT, TSREQ+TSRST
MOV        CX, 100
LOOP       LOOP
PUSH       $
PUSH       PCNT, TSREQ
MOV        CX, 1000
LOOP       LOOP
POP        CX
POP        AX
JMP        JMP

HDCMD0:
PUTP1      PSDATA, 0FFh
PUTP1      PCNT, TSREQ
HDCMD0:
GETP       PCNT
TEST       AL, TSREQ
JNE        JE
POP        JE
JMP        JMP

HDCMD1:
TEST       AL, TSREQ
JNE        JE
POP        JE
JMP        JMP

HDCMD1:
MOV        DI, CMDADR
CALL       MANTFR
JNC        JMP
HDCMD1:
JMP        JMP

HDCMD1:
GETP       PCNT
TEST       AL, TSREQ
JNE        JE
POP        JE
JMP        JMP

```

	JNE	HDCMTRF	
	TEST	AL, TSMG+TSCD+TS10	: ステータスフューズの場合もあるので
	JE	HDCMSTS	: チェックする
	MOV	DI, BUFADDR	
	MOV	ES, BUFADDR+2	: DI ← データバッファアドレス
	TFMODE	ES, BUFADDR+2	: E ← データバッファセグメント
	TEST	TFMODE	
	JE	HDCMAN	: プライント転送モード?
	CALL	BLINDTR	: プライント転送
	JMP	HDCM1	
	CALL	MANTR	: マニュアル転送
	PUSH	CS	
	POP	ES	
	GETP	PSSNS	
	TEST	AL, T3REQ	: ステータスフューズ
	JNE	HDCMSTS	: REQ が 1 になるまで待つ
	TEST	AL, TSMG+TSCD+TS10	
	JNE	HDCMSTS	
	MOV	BX, CMDADDR	
	LEA	DI, [BX], STAT	
	CALL	MANTR	: ステータスバイトを得る
	GETP	PSSNS	: メッセージインフューズ
	TEST	AL, T3REQ	: REQ が 1 になるまで待つ
	JNE	HDCMDES	
	AND	AL, TSMG+TSCD	
	CMP	AL, TSMG	
	JNE	HDCMDES	
	PUSH	CS	
	POP	ES	
	MOV	BX, CMDADDR	
	LEA	DI, [BX], MESS	
	CALL	MANTR	: メッセージバイトを得る
	GETP	PSSNS	: バスフリーフューズ
	TEST	AL, T3REQ	: ビジーが 0 になるまで待つ
	JNE	HDCMDF	
	MOV	DI, BUFADDR	: ディスコネクト
	MOV	ES, BUFADDR+2	: バッファアドレスの復帰
	CLC		: 正常終了
	RET		
	MOV	DI, BUFADDR	
	MOV	ES, BUFADDR+2	: 異常終了
	STC		
	RET		
	PAGE		
			プライント転送ルーティン (ハンドシェイクを行わない)
			本日は、やりたくないのだが、このハードウェアではどうでもしないと
			本当は、コンピュータが持っているというべきで、プログラムの使用
			現在、バスバッファアドレスは ES:DI
	BLINDTR:	DI	
		SI	
		BX, CMDADDR	: SCSI リクエストヘッダから
		AL, 0	

	Mov	AL,[BX],BLOCK	:転送すべきバイト数/5 12を得て ;それをCXレジスタにセットして
	Mov	CX,AX	
	Mov	AL,BYTE PTR SS:[Z]	:セクタサイズが5 12の場合は
	JE	BLIFR05	:ループカウンタを2倍にする
	SHL	CX,1	
BLIFR05:	Mov	BX,PSDATA	
	PUSH	DS	
	PUSH	ES	:DS-ES
	POP	CLD	
	PUTP1	PSDATA,0fffh	:Data Lineをクリア
BLIFR10:	GETP	PSNIS	
	TEST	AL,TSRQO	: R E Qがtrueになるまで待つ
	JNE	BLIFR10	
BLIFR15:	TEST	AL,TSIO	:データイン/アウトをチェックする
	JNE	BLIOUT0	
BLIIN0:	PUTP1	PSCNT,TSIOS+TSIO0IR+TSATN	:Data portを入力モードに設定
BLIIN1:	PUSH	CX	
	Mov	CX,256	:256bytesずつblind転送
BLIIN2:	GETP	PSNIS	
	TEST	AL,TSREQ	: R E Qがtrueになるまで待つ
	JNE	BLIIN2	
	TEST	AL,TSQO	:フェーズが変わってはいればエラー
	JE	BLIFERR	
BLIIN4:	Mov	DX,PSNIS	
	IN	AL,DX	
	TEST	AL,TSREQ	: R E Qがtrueになるまで待つ
	JNE	BLIIN4	
BLIIN5:	DEC	DX	:DX←PSDATA
	IN	AL,DX	:データを取得
	NOT	AL	
	STOSB		
	Mov	AX,(TSIOS+TSIO0IR+TSATN)*256+TSIOS+TSIO0IR+TSATN+TSACK	
	Mov	DX,PSCNT	
	OUT	DX,AL	:Set ACKビット
	Mov	DX,PSCNT	
	Mov	AL,AH	
	OUT	DX,AL	:Reset ACKビット
	Mov	DX,PSNIS	
	LOOP	BLIIN4	
	POP	CX	
	LOOP	BLIIN1	
	PUTP1	PSCNT,TSATN	:Data portを出力モードに設定
	JMP	BLIFR20	
BLIOUT0:	Mov	SI,D1	
BLIOUT1:	PUSH	CX	
	Mov	CX,511	
	Mov	DX,PSNIS	
BLIOUT2:	IN	AL,DX	: R E Qがtrueになるまで待つ
	TEST	AL,TSREQ	

	JNE	BLIOUT2	
	TEST JE	AL, TSCD	: フェーズが変わっていればエラー
		BLITFRERO	: 最初の1バイトだけハンドシェイクする
	LODSB		
	NOT PUTP	AL	
		PSDATA	
BLIOUT3:	PUTPI MOV	PSCNT, TSATN+TSACK DX, PSSNS	: Set ACKビット
	IN TEST JE	AL, DX	: REQビットがfalseになるまで待つ
		AL, TREQ	
		BLIOUT3	
	PUTPI	PSCNT, TSATN	: Reset ACKビット
BLIOUT4:	MOV	DX, PSSNS	
	IN TEST JNE	AL, DX	: REQがtrueになるまで待つ
		AL, TREQ	
		BLIOUT4	
BLIOUT5:	LODSB		: データアクトの場合
	NOT DEC	AL	: DX←PSDATA
	INC	AL	: データを送出
	MOV	AX, (TSATN)*256+TSATN+TSACK	
	MOV	DX, PSCNT	
	OUT	DX, AL	: Set ACKビット
	MOV	AL, AH	
	OUT	DX, AL	: Reset ACKビット
	MOV LOOP	DX, PSSNS	
		BLIOUT4	
	POP LOOP	CX	
		BLIOUT1	
BLITRZ0:	POP JMP	DS	
		BLITFREXIT	
BLITFRER:	POP PUTPI	CX	
	POP POP	PSDATA, offh	: Data Lineをクリア
	POP STC	SI	: 異常終了
	RET	D1	
BLITFREXIT:	PUTPI POP	PSDATA, offh	
	POP POP	SI	: Data Lineをクリア
	RET	D1	: 正常終了
BLITFRERO:	POP PUTPI	CX	
	POP POP	PSDATA, offh	
	JMP	DS	: Data Lineをクリア
PAGE		BLITFREXIT	
:			
:			
:			
:			
MANTFR:	PUSH PUTPI	D1	
		PSDATA, offh	
MANTFRIO:	GETP TEST	PSSNS	
		AL, TREQ	: REQがtrueになるまで待つ

アニュアム転送ルーティン  
 フェーズ変化するまで転送する  
 バッファアドレスはE:S:D1



＜リスト9＞ SCSI.ASM (つづき)

MANTFR20:	JNE AND MOV	MANTFR10 AL, TMSGD+TSCD+TS10 TMRPHA, AL	:現在のバスフューズをストア
MAIN:	TEST JNE	AL, TS10 MANOUT	:!ln+Outかをチェック
	PUTP1 MOV LOOP	PSCNT, TS10ST+TS0DIR+TSATN CX, 100	:DATA IN フューズ :Data portを入力モードに設定 :Data setup delay
	GETP NOT MOV	PSDATA AL ES: [DI], AL	:Dataを得る
	PUTP1	PSCNT, TSATN ;Data portを出力モードに設定	
MANOUT:	JMP MOV NOT PUTP MOV LOOP	SETACK AL, ES: [DI] PSDATA CX, 100	:DATA OUT フューズ :Dataを送出 :Data setup delay
SETACK:	PUTP1	PSCNT, TSATN+TSACK	:Set ACKビット
WAITREQ:	GETP TEST JE	PSHS TEST AL, TSREQ WAITREQ	:REQビットがfalseになるまで待つ
	PUTP1	PSCNT, TSATN ;Reset ACKビット	
	INC	DI	:バッファのポインタを進める
MANFR30:	GETP TEST JNE JMP	PSHS TEST AL, TSREQ MANFR40 AL, TBSVY MANTFRXIT MANTFR30	:次のREQが来るまで待つ :Bus Free Phaseなら終了
MANFR40:	AND JNE	AL, TMSGD+TSCD+TS10 TMRPHA MANTFR20	:今度のフューズを得る :今までと同じ? :なまらう1バイト
MANTFRXIT:	PUTP1 POP CLC RET	PSDATA, 0fth DI	:Data Lineをクリア :終了
SC11:	PUSH PUSH PUSH PUTP1 MOV LOOP PUTP1 MOV PUSH MOV	AX PSH1 DX PSCNT, TSATN+TSRST CX, 100 PSCNT, TSATN CX, WORD PTR RSTWT CX, 55000	:リセット信号をセット :リセット信号をクリア :指定秒数待つ

[illegible]

MOV SCS1, SCS1_00H	SCS1, SCS1_00H	: 1 セクタだけ読み込む
MOV MOV SCS1, BLACK_01H	SCS1, BLACK_01H	
MOV MOV SCS1, EXT1_00H	SCS1, EXT1_00H	
AM, SCS1	AM, SCS1	: ES ← CS
ES	ES	
DI, DISCT	DI, DISCT	: マニュアル転送を選択
TFMODE, 00H	TFMODE, 00H	
DIERR	DIERR	: ステータスチェック
SCS1, STAT_00H	SCS1, STAT_00H	: ノーエラー
DIEX1	DIEX1	: センサデータを取得
REASSENS	REASSENS	
DIERR	DIERR	: リカバードエラーは無視する
AL, BYTE PTR SENS DAT+2	AL, BYTE PTR SENS DAT+2	: リセットコンディションだともう1度
AL, 01H	AL, 01H	
DIEX1	DIEX1	
AL, 06H	AL, 06H	
DINIT1	DINIT1	
DX, MSGERR	DX, MSGERR	: 失敗のメッセージを表示
AH, 09H	AH, 09H	
21H	21H	
DS	DS	: リクエストヘッダアドレスを得る
AL, 0	AL, 0	: フライプ数 ← 0
DI, [PTRSAV]	DI, [PTRSAV]	
[BX, MEDIA], AL	[BX, MEDIA], AL	
WORD PTR [BX, TRANS], OFFSET DSK_INIT	WORD PTR [BX, TRANS], OFFSET DSK_INIT	
WORD PTR [BX, TRANS+2], CS	WORD PTR [BX, TRANS+2], CS	
WORD PTR [BX, COUNT], OFFSET INIT_TBL	WORD PTR [BX, COUNT], OFFSET INIT_TBL	
WORD PTR [BX, COUNT+2], CS	WORD PTR [BX, COUNT+2], CS	
POP DS	DS	: B P B 配列のポインタ
STC	AL, ESNOTRDY	
MOV ERR_EXIT	ERR_EXIT	: エラーコードのセット
DX, MSGCMP	DX, MSGCMP	: 成功のメッセージを表示
AH, 09H	AH, 09H	
21H	21H	
CH, 0	CH, 0	: D D セクタから、パーティションの
CL, BYTE PTR DISCT, DRWNUM	CL, BYTE PTR DISCT, DRWNUM	: 数を得る
DL, BYTE PTR STDRDY	DL, BYTE PTR STDRDY	
DI, 1	DI, 1	
BX, DSKRPT0+9	BX, DSKRPT0+9	
DX	DX	: ドライブ番号の表示
PUSH AH, 02H	AH, 02H	
INT 21H	21H	
MOV DL, :	DL, :	
MOV AL, 1	AL, 1	
MOV AL, BYTE PTR [BX]	AL, BYTE PTR [BX]	
ADD ADD AL, 02H	AL, 02H	
SIR AL, 1	AL, 1	
SIR AND AL, 07EH	AL, 07EH	
MOV AH, 0	AH, 0	
PUSH BX, CAPTABLE	BX, CAPTABLE	
INT 21H	21H	
MOV DL, BYTE PTR [BX]	DL, BYTE PTR [BX]	
MOV AH, 02H	AH, 02H	
INT 21H	21H	
INT DL, BYTE PTR [BX+1]	DL, BYTE PTR [BX+1]	
INT 21H	21H	
POP BX	BX	

[illegible]



クリスト10> EASYHARD.INC

[illegible]







```

} else {
    bpb[ i ].allsiz = 1;
}
bpb[ i ].fatno = vol / bpb[ i ].allsiz * 2 / 512 + 1;
}
putchar( '\n' );
for( i = 0; i < patno; i++ ) {
    printf( "パーティション %3d: %sセクタ(%dバイト) %m", i + 1, bpb[ i ].sectvol, 512L * bpb[ i ].sectvol );
    printf( "残りセクタ: %d\n", restset );
    printf( "これで良いですか? (Y/N) ");
    scanf( "%s", buf );
    while( #buf != 'y' && #buf != 'Y' );
}
printf( "\nFATとディレクトリの初期化を行います。 %m", '\n' );
restset = 3L;
for( i = 0; i < patno; i++ ) {
    printf( "パーティション %3d %m", i + 1 );
    for( j = 0; j < 512; j++ ) {
        buf[ j ] = 0;
    }
    for( l = 1; l < 2L * bpb[ i ].fatno + 32L; l++ ) {
        write( id, lun, restset + l, buf );
        buf[ 0 ] = 0xff;
        buf[ 1 ] = 0xff;
        buf[ 2 ] = 0xff;
        write( id, lun, restset + lL + bpb[ i ].fatno, buf );
        write( id, lun, restset + lL + bpb[ i ].fatno, buf );
        memcpy( &buf[ 3 ], "EHD V1.2", 8 );
        memcpy( &buf[ 11 ], &bpb[ i ], sizeof( bpb[ i ] ) );
        write( id, lun, restset, buf );
        restset += bpb[ i ].sectvol;
    }
}
restset = 3L;
ddsect.pn = patno;
for( i = 0; i < 12; i++ ) {
    ddsect.sectofs[ i ] = restset;
    restset += bpb[ i ].sectvol;
}
memcpy( &buf[ 0 ], &ddsect, sizeof( ddsect ) );
for( l = 0; l < 12; l++ ) {
    memcpy( &buf[ 64 + l * 16 ], &bpb[ i ], sizeof( bpb[ i ] ) );
}
write( id, lun, 0L, buf );
}

/* Verify */
verify( id, lun, start, range, buf )
int id, lun;
long start, range;
unsigned char #buf;
{
    unsigned char v[ 10 ], sensbuff[ 128 ];
    int sens;
    v[ 0 ] = 0x2f;
    v[ 1 ] = lun << 5;
    v[ 6 ] = v[ 9 ] = 0x00;
    v[ 2 ] = { start >> 24 } & 0x000000ffL;
    v[ 3 ] = { start >> 16 } & 0x000000ffL;
    v[ 4 ] = { start >> 8 } & 0x000000ffL;
    v[ 5 ] = { start >> 0 } & 0x000000ffL;
    v[ 6 ] = { range >> 24 } & 0x000000ffL;
    v[ 7 ] = { range >> 16 } & 0x000000ffL;
    v[ 8 ] = { range >> 8 } & 0x000000ffL;
    v[ 9 ] = { range >> 0 } & 0x000000ffL;
    daa = 0; /* not DMA mode */
    if( hdcmd( id, v, buf ) ) {
        sens = requestSense( id, lun, buf );
        if( sens == 1 || sens == 3 ) {
            return( 1 );
        }
    } else {
        printSense( buf );
    }
}

```

```

}
return( 0 );
}

/* Reassign block */
reassign( id, lun, badblock )
int id, lun;
long badblock;
{
    unsigned char v[ 6 ], buf[ 128 ];
    printf( "\nブロック %d が不良です。交替処理を行います。 %m", badblock );
    v[ 0 ] = 0x0f;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    buf[ 0 ] = 0;
    buf[ 1 ] = badblock >> 24;
    buf[ 2 ] = badblock >> 16;
    buf[ 3 ] = badblock >> 8;
    buf[ 4 ] = badblock >> 0;
    buf[ 5 ] = { badblock >> 24 } & 0x000000ffL;
    buf[ 6 ] = { badblock >> 16 } & 0x000000ffL;
    buf[ 7 ] = { badblock >> 8 } & 0x000000ffL;
    buf[ 8 ] = { badblock >> 0 } & 0x000000ffL;
    daa = 0; /* not DMA mode */
    if( hdcmd( id, v, buf ) ) {
        sens = requestSense( id, lun, buf );
        if( sens == 1 || sens == 3 ) {
            printSense( buf );
        }
    }
}

/* Test unit ready */
testUnitReady( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], buf[ 80 ];
    int i;
    v[ 0 ] = C.TESTUDY;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    daa = 0; /* not DMA mode */
    return( hdcmd( id, v, buf ) );
}

/* Mode select */
modeSelect( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], sensbuff[ 256 ];
    int i;
    v[ 0 ] = C.MODESEL;
    v[ 1 ] = { lun << 5 } + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    for( i = 0; i < 12; i++ ) {
        buf[ i ] = 0x00;
    }
    buf[ 3 ] = 0x08;
    buf[ 10 ] = { ( sectsiz == 256 ) ? 0x01 : 0x02 }; /* 256 or 512 byte / sector */
    buf[ 12 ] = 0x01;
    buf[ 13 ] = 0x04;
    buf[ 14 ] = 0x04;
    buf[ 15 ] = 0x08;
    buf[ 16 ] = 0x08;
    buf[ 17 ] = 0x00;
    buf[ 18 ] = 0x00;
    buf[ 19 ] = 0x00;
}

```



```

/*
dma = 0; /* not DMA mode */
if( hcmd( id, v, buf ) )
    if( requestense( id, lun, sensbuf ) == 6 ) {
        /* reset condition */
        hcmd( id, v, buf );
    }
    printsens( sensbuf );
}

/* read capacity */
readcapacity( id, lun, buf )
int id, lun;
{
    unsigned char v[ 10 ], sensbuf[ 256 ];

    v[ 0 ] = C.RDCAP;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00; /*logical block address is 0 */
    v[ 6 ] = v[ 7 ] = 0x00; /* PHL bit, false */
    v[ 8 ] = 0x00; /* flag and link, false */
    v[ 9 ] = 0x00;

    dma = 0; /* not DMA mode */
    if( hcmd( id, v, buf ) )
        if( requestense( id, lun, sensbuf ) == 6 ) {
            /* reset condition */
            hcmd( id, v, buf );
        }
        printsens( sensbuf );
    }

/* Format Unit */
format( id, lun, inlv )
int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];

    v[ 0 ] = C.FMTDRV;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    v[ 6 ] = inlv;
    dma = 0; /* not DMA mode */
    if( hcmd( id, v, sensbuf ) )
        if( requestense( id, lun, sensbuf ) == 6 ) {
            /* reset condition */
            hcmd( id, v, sensbuf );
        }
        printsens( sensbuf );
    }

/* Mode select */
modeselectfirst( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], buf[ 128 ], sensbuf[ 256 ];
    int i, cyl, hd, wp, rec;

    do {
        printf( "v[0] このドライブの、シリンダ数はいくつですか？ " );
        scanf( "%d", &cyl );
        printf( "v[1] ヘッド 数はいくつですか？ " );
        scanf( "%d", &hd );
        printf( "v[2] 書き込みは相補正 (タイプ1コンベ) を始めますか？ (不要な場合は、0) " );
        printf( "v[3] シリンダ数はいくつですか？ (不要な場合は、0) " );
        printf( "v[4] シリンダ数と同じ数 " );
        scanf( "%d", &wp );
    }

```

```

printf( "v[5] 書き込み相補正 (リゼースドライブタイプ1コンベ) " );
printf( "v[6] シリンダ数はいくつですか？ (不要な場合は、0) " );
scanf( "%d", &rec );
printf( "v[7] これより、v[0] のシリンダ数と同じ数 " );
scanf( "%s", buf );
}while( *buf != '\0' );

for( i = 0; i < 127; i++ ) {
    buf[ i ] = 0;
}

v[ 0 ] = C.MODESEL;
v[ 1 ] = ( lun << 5 ) * 0x10;
v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00; /* parameter byte no. */
v[ 6 ] = 36; /* block header */
buf[ 0 ] = 0x00; /* block descriptor */
buf[ 1 ] = 0x00;
buf[ 2 ] = 0x00;
buf[ 3 ] = 0x08;
buf[ 4 ] = 0x00;
buf[ 5 ] = 0x00;
buf[ 6 ] = 0x00;
buf[ 7 ] = 0x00;
buf[ 8 ] = 0x00;
buf[ 9 ] = 0x00;
buf[ 10 ] = ( ( sectsiz == 256 ) ? 0x01: 0x02 ); /* 256 or 512 byte / sector */
buf[ 11 ] = 0x00;
buf[ 12 ] = 0x04; /* page code 4 */
buf[ 13 ] = 0x00; /* parameter byte no. */
buf[ 14 ] = 0x00; /* No. of cylinders */
buf[ 15 ] = cyl >> 8;
buf[ 16 ] = cyl & 0x00ff;
buf[ 17 ] = hd;
buf[ 18 ] = 0x00; /* No. of heads */
buf[ 19 ] = wp >> 8;
buf[ 20 ] = wp & 0x00ff; /* write precomp start cylinder */
buf[ 21 ] = rec >> 8;
buf[ 22 ] = rec & 0x00ff; /* reduced write current start cylinder */
buf[ 23 ] = rec & 0x00ff;
buf[ 24 ] = 0x03; /* page code 3 */
buf[ 25 ] = 0x16;
buf[ 26 ] = 0x00;
buf[ 27 ] = 0x00;
buf[ 28 ] = 0x00;
buf[ 29 ] = 0x00;
buf[ 30 ] = 0x00;
buf[ 31 ] = 0x00;
buf[ 32 ] = 0x00;
buf[ 33 ] = 0x00;
buf[ 34 ] = 0x00;
buf[ 35 ] = 0x00;
buf[ 36 ] = 0x00;
buf[ 37 ] = 0x00;
buf[ 38 ] = 0x00;
buf[ 39 ] = 0x00;
buf[ 40 ] = 0x00;
buf[ 41 ] = 0x01; /* track skew factor */
buf[ 42 ] = 0x00;
buf[ 43 ] = 0x00;
buf[ 44 ] = 0x00;
buf[ 45 ] = 0x00;
buf[ 46 ] = 0x00;
buf[ 47 ] = 0x00;
buf[ 48 ] = 0x01; /* page code 1 */
buf[ 49 ] = 0x06;
buf[ 50 ] = 0x24;
buf[ 51 ] = 0x00;
buf[ 52 ] = 0x00;
buf[ 53 ] = 0x00;
buf[ 54 ] = 0x00;
buf[ 55 ] = 0x00;

```

```

/* Write 1 Sector */
write( id, lun, lsn, buf )
int id, lun; /* logical sector number */
long lsn; /* data buffer */
unsigned char *buf;

{
    unsigned char v[ 6 ];
    unsigned char sensbuf[ 256 ];
    if( sectsz == 256 ) {
        lsn = 2L;
    }
    v[ 0 ] = C.WRTSEC;
    v[ 4 ] = ( ( sectsz == 256 ) ? 2 : 1 );
    v[ 5 ] = 0x00;
    v[ 1 ] = ( lun << 5 ) + ( lsn >> 16 ) & 0x00ff;
    v[ 2 ] = ( lsn >> 8 ) & 0x00ff;
    v[ 3 ] = lsn & 0x00ff;
    dma = 0; /* write 256 bytes */
    sectmode = 0x0b88;
    if( hcmd( id, v, buf ) ) {
        if( request sense( id, lun, sensbuf ) == 6 ) {
            /* reset condition */
            hcmd( id, v, buf );
        }
        else {
            printsens( sensbuf );
        }
    }

    request sense( id, lun, buf )
    int id, lun;
    unsigned char *buf;

    /* check error, do request sense */
    unsigned char v[ 6 ];

    v[ 0 ] = C.REQSEN;
    v[ 1 ] = lun << 5; /* set lun */
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = v[ 5 ];
    dma = 0x0055;
    hcmd( id, v, buf );
    return( buf[ 2 ] );

    printsens( buf )
    unsigned char *buf;

    printf( "Sense key = %02X, Extend sense code = %02X\n", buf[ 2 ], buf[ 12 ] );
    dump( buf );
    exit( 99 );
}

/* output command to HDD */
hcmd( id, sc, buf )
int id;
unsigned char *sc, *buf;

{
    int treg; /* temp. reg. */
    int idb;

    idb = 0x0001 << id;
    /* selection phase */
    printf( "Selection.Vn" ); /* set target & init. ID */
    outp( P.DATA, ( INITBIT+idb ) ); /* selection bit true */
    outp( P.CNT, ATNSEL );
    for( treg = 0; treg < 1000; treg++ ) {
        /* wait responding time ( 200us max. ) */
    }
}

```

```

if( hcmd( id, v, buf ) ) {
    if( request sense( id, lun, sensbuf ) == 6 ) {
        /* reset condition */
        hcmd( id, v, buf );
    }
    else {
        printsens( sensbuf );
    }
}

/* Format Unit */
format( id, lun, inv )
int id, lun, inv;

{
    unsigned char v[ 6 ]; sensbuf[ 256 ];
    int dfno, i;
    struct {
        int cyl;
        int hd;
        int diff;
    } defect[ 1024 ];
    char buf[ 1024 ];
    printf( "フォーマットの値はいくつですか？ " );
    scanf( "%d", &dfno );
    do {
        for( i = 0; i < dfno; i++ ) {
            printf( "インデックスは？ シリンダは？ ヘッドは？ " );
            scanf( "%d %d %d", &( defect[ i ].cyl ), &( defect[ i ].hd ), &( defect[ i ].diff ) );
            printf( "インデックスからの距離(バイト)は？ " );
            scanf( "%d", &( defect[ i ].diff ) );
        }
        putchar( '\n' );
        for( i = 0; i < dfno; i++ ) {
            printf( "インデックスは？ シリンダは？ ヘッドは？ " );
            scanf( "%d %d %d", &( defect[ i ].cyl ), &( defect[ i ].hd ), &( defect[ i ].diff ) );
            printf( "インデックスからの距離(バイト)は？ (V/N) " );
            scanf( "%s", buf );
            while( *buf != '\n' && *buf != '\0' );
        }

        v[ 0 ] = C.FMTDRV;
        v[ 1 ] = ( lun << 5 ) + 0x1c;
        v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
        v[ 4 ] = inv;

        buf[ 0 ] = 0x00; /* create p-list */
        buf[ 1 ] = 0x0f;
        buf[ 2 ] = ( dfno * 8 ) >> 8;
        buf[ 3 ] = ( dfno * 8 ) & 0x00ff;

        ptr = &( buf[ 4 ] );
        for( i = 0; i < dfno; i++ ) {
            *ptr++ = defect[ i ].cyl >> 8;
            *ptr++ = defect[ i ].cyl & 0x00ff;
            *ptr++ = defect[ i ].hd >> 8;
            *ptr++ = defect[ i ].hd & 0x00ff;
            *ptr++ = 0;
            *ptr++ = 16;
            *ptr++ = 0;
        }
        dma = 0; /* not DMA mode */
        printf( "フォーマットを行っています。2～5分程度お待ち下さい。Vn" );
        if( hcmd( id, v, buf ) ) {
            if( request sense( id, lun, sensbuf ) == 6 ) {
                /* reset condition */
                hcmd( id, v, buf );
            }
            else {
                printsens( sensbuf );
            }
        }
    }
}

```



```

if( inp( P_SNS ) & BSY ) { /* if BSY is false */
    printf( "couldn't select device.\n" );
    exit( 99 );
}
outp( P_DATA, 0x0f ); /* clear data line */
outp( P_CMD, 0x00 ); /* selection complete. reset selection bit. */
/* command phase */
printf( "Command.\n" );
do {
    while( ( treg = inp( P_SNS ) ) & REQ ) {
        /* wait while REQ = false */
        treg &= CD;
        while( treg ); /* wait until command phase */
        printf( "sc" ); /* output command data */
        /* data transfer phase */
        printf( "Data Transfer.\n" );
        while( ( treg = inp( P_SNS ) ) & REQ ) {
            /* wait while REQ = false */
            if( ( treg & ( MSG + CD + IO ) ) != 0 ) { /* status phase? (MSG=f, CD, IO=t) */
                /* no data phase. */
                printf( "buf" );
            }
            /* status phase */
            printf( "Status.\n" );
            do {
                while( ( treg = inp( P_SNS ) ) & REQ ) {
                    /* wait while REQ = false */
                }
                while( ( treg & ( MSG + CD + IO ) ) != 0 ); /* wait until MSG=f, CD, IO=t */
                tfsbuf( &tbuf ); /* get status byte */
            }
            /* message phase */
            printf( "Message.\n" );
            do {
                while( ( treg = inp( P_SNS ) ) & REQ ) {
                    /* wait until REQ = true */
                }
                treg &= ( MSG + CD );
                while( treg != MSG );
                tfsbuf( &tbuf ); /* get message byte */
            }
            /* bus free phase */
            printf( "Bus free.\n" );
            while( ( treg = inp( P_SNS ) & BSY ) ) {
                /* wait until BUSY = false */
            }
            /* disconnect */
            treg = tfsbuf;
            if( treg == 0x01 ) {
                /* parity error */
                printf( "parity error occurred.\n" );
                exit( 99 );
            }
            while( treg == 0x02 ) {
                /* check condition */
                return( 1 );
            }
            else {
                return( treg );
            }
        }
    }
}

/* manual Data transfer routine */
printf( "buf" );
unsigned char tbuf;
int treg; /* temporary register */
int tphase; /* initial phase storage */
while( ( treg = inp( P_SNS ) ) & REQ ) {
    /* wait until REQ = true */
    tphase = treg & ( MSG + CD + IO );
    do {

```

```

tfsbuf( tbuf );
while( ( treg = inp( P_SNS ) ) & REQ ) {
    /* wait until REQ = true */
    treg &= ( MSG + CD + IO );
    while( treg == tphase ); /* repeat until phase change */
    return( 0 );
}

/* sub routine: 1 byte transfer */
tfsbuf( tbuf );
unsigned char tbuf;
int treg; /* temp. reg. */
outp( P_DATA, 0x0f ); /* clear data line */
while( ( treg = inp( P_SNS ) ) & REQ ) {
    /* wait while REQ = false */
    if( ( treg & IO ) == 0 ) {
        outp( P_CMD, 0x00 + tphase );
        /* I/O IN */
        tbuf = inp( P_DATA );
        outp( P_CMD, ATN );
    }
    else {
        /* I/O OUT */
        outp( P_DATA, tbuf );
        outp( P_CMD, ATN + ACK );
        while( ( treg = inp( P_SNS ) ) & REQ ) {
            /* wait until REQ = false */
        }
        outp( P_CMD, ATN );
        /* reset ACK bit */
        outp( P_DATA, 0x0f ); /* clear data line */
    }
}

dump( tbuf );
unsigned char tbuf;
int i, j;
unsigned d;
printf( "00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0123456789ABCDEF\n" );
for( i = 0; i < 256; i += 16 ) {
    printf( "%04X", i );
    for( j = 0; j < 16; j++ ) {
        printf( "%02X", tbuf[ i + j ] );
    }
    for( j = 0; j < 16; j++ ) {
        d = tbuf[ i + j ];
        if( ( d >= 0x10 ) && ( d < 0x7f ) ) {
            putchar( d );
        }
        else {
            putchar( '.' );
        }
    }
    putchar( '\n' );
}
}

```

```

/* Format program for Easyllard and SCSI-CCS HDD
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <easyhard.h>

int setsiz;

int main()
{
    unsigned char buf[ 512 ];
    int i, j, id, lun, int, patno;
    long i, maxset, stt;

    printf( "HDD initialize utility V1.3 for Easyllard and SCSI-CCS HDD\n" );
    printf( "Copyright by Tsuru-Zoh May.06.1990\n" );
    printf( "Yn *** Warning! *** all data on HDD will be lost!\nYn" );
    printf( "Enter the SCSI-ID to format (0 to 9): " );
    scanf( "%d", &id );
    printf( "Enter the SCSI-LUN to format (0 to 9): " );
    scanf( "%d", &lun );
    reset( id, lun );

    printf( "Yn Physical format desired? (Y/N) " );
    scanf( "%s", buf );
    if( #buf == 'Y' || #buf == 'y' ) {
        physformat( id, lun );
    }

    readcapacity( id, lun, buf );
    if( ( buf[ 6 ] != 1 ) && ( buf[ 6 ] != 2 ) ) {
        printf( "Physical sector size must be 256 or 512.\n" );
        exit( 1 );
    }
    if( setsiz == 0 ) {
        setsiz = buf[ 6 ] * 256;
    }
    if( ( buf[ 6 ] * 256 ) != setsiz ) {
        printf( "Selected Sector size : %d Physical Sector size : %Yn",
            setsiz, buf[ 10 ] * 256 );
        setsiz = buf[ 10 ] * 256;
    }
    printf( "Ignore and continue? (Y/N) " );
    scanf( "%s", buf );
    if( ( #buf != 'Y' ) && ( #buf != 'y' ) ) {
        exit( 1 );
    }

    maxset = 65536L * buf[ 1 ] + 256L * buf[ 2 ] + buf[ 3 ] + 1L;
    printf( "Media Verify desired? (Y/N) " );
    scanf( "%s", buf );
    if( #buf == 'Y' || #buf == 'y' ) {
        verifyunit( id, lun, maxset );
    }
    makepart( id, lun, maxset );
    printf( "Format complete.Yn" );
    printf( "Add SCSI.SYS to config.sys, and reboot please.Yn" );
}

/* SCSI BUS-Reset
reset( id, lun )
int id, lun;
{
    long i, t;

    printf( "nSCSI BUS-Reset executing. Wait 10 sec.Yn" );
    outp( P_CNT, ATN );
    for( i = 0L; i < 300L; i++ ) {

```

```

        outp( P_CNT, ATN );
        for( i = 0L; i < 100000L; i++ ) {
            i = time( (long*)0 );
            while( ( t = (time( (long*)0 ) - 1 ) ) < 10L ) {
                printf( "%dYr. t", t );
            }
            printf( "Yn" );
        }
        /* physical format */
        physformat( id, lun );
        int id, lun;

        int int;
        char buf[ 32 ];

        do {
            printf( "Enter the sector size ? (256 or 512) " );
            scanf( "%d", &setsiz );
            while( ( setsiz != 256 ) && ( setsiz != 512 ) );
            printf( "Enter the Interleave factor? " );
            scanf( "%d", &int );

            printf( "Is this the first time to format this HDD? (Y/N) " );
            scanf( "%s", buf );
            if( #buf == 'Y' ) {
                modefirst( id, lun );
            }
            else {
                formatfirst( id, lun, int );
            }
            modefirst( id, lun );
            format( id, lun, int );
        } while( "YnPhysical format complete.Yn" );

        verifyunit( id, lun, maxblock );
        int id, lun;
        long maxblock;

        long i, lt, le, badblock;
        unsigned char buf[ 256 ];
        int times;

        printf( "YnNo. of Sector is %d.Yn", maxblock );
        printf( "How many times do you want to verify the Media?" );
        scanf( "%d", &times );
        putchar( 'Yn' );
        for( i = 0; i < times; i++ ) {
            printf( "Media verify %d times.Yn", i + 1 );
            for( lt = 0L; lt < maxblock; lt += 1024L ) {
                printf( "Block %dYr. lt", lt );
                lt = 1;
                while( ( lt + 1024L ) > maxblock ) { maxblock = lt + 1024L;
                    while( verify( id, lun, lt, le, buf ) ) {
                        badblock = 65536L * buf[ 4 ] + 256L * buf[ 5 ] + buf[ 6 ];
                        reassign( id, lun, badblock );
                        le = badblock;
                        lt = 1;
                        if( lt == ( 1 + 1024L ) ) {
                            break;
                        }
                    }
                }
            }
            printf( "YnMedia Verify complete.Yn" );
        }
        makepart( id, lun, maxset );
        int id, lun;
        long maxset;

        static struct {
            int pri;
            long sectors[ 20 ];
        } ddsct;
        static struct {

```



[illegible]

```

}

restsect = 3L;
ddsect, pn = ratno;
for( i = 0; i < 12; i++ ){
    ddsect, sctof[ i ] = restsect;
    restsect += bpb[ i ].sctvol;
}
mency[ &( bbuf[ 0 ] ), &ddsect, sizeof( ddsect ) ];
for( i = 0; i < 12; i++ ){
    mency[ &( bbuf[ 64 + i * 16 ] ), &( bpb[ i ] ), sizeof( bpb[ i ] ) ];
}
fwrite( id, lun, 0L, buf );
}

/* Verify */
verify( id, lun, start, range, buf )
int id, lun;
long start, range;
unsigned char *buf;
{
    unsigned char v[ 10 ], sensbuff[ 128 ];
    int sens;

    v[ 0 ] = 0x2f;
    v[ 1 ] = lun << 5;
    v[ 6 ] = v[ 9 ] = 0x00;

    v[ 2 ] = { start >> 24 } & 0x000000ffL;
    v[ 3 ] = { start >> 16 } & 0x000000ffL;
    v[ 4 ] = { start >> 8 } & 0x000000ffL;
    v[ 5 ] = { start >> 0 } & 0x000000ffL;
    v[ 6 ] = { range >> 8 } & 0x000000ffL;
    v[ 7 ] = { range >> 0 } & 0x000000ffL;
    v[ 8 ] = range & 0x000000ffL;

    dma = 0; /* not DMA mode */
    if( bdcmd( id, v, buf ) ){
        sens = requestsens( id, lun, buf );
        if( sens == 1 ) sens = 3;
        return( 1 );
    }
    else{
        printsens( buf );
    }
}

return( 0 );
}

/* Reassign block */
reassign( id, lun, badblock )
int id, lun;
long badblock;
{
    unsigned char v[ 6 ], buf[ 128 ];

    printf( "Block %d is Bad. Reassign it.\n", badblock );

    v[ 0 ] = 0x07;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    buf[ 0 ] = { badblock >> 24 } & 0x000000ffL;
    buf[ 1 ] = { badblock >> 16 } & 0x000000ffL;
    buf[ 2 ] = { badblock >> 8 } & 0x000000ffL;
    buf[ 3 ] = { badblock >> 0 } & 0x000000ffL;
    buf[ 4 ] = { badblock >> 24 } & 0x000000ffL;
    buf[ 5 ] = { badblock >> 16 } & 0x000000ffL;
    buf[ 6 ] = { badblock >> 8 } & 0x000000ffL;
    buf[ 7 ] = badblock & 0x000000ffL;

    dma = 0; /* not DMA mode */
    if( bdcmd( id, v, buf ) ){
        sens = requestsens( id, lun, buf );
        printsens( buf );
    }
}

/* Test Unit ready */
testready( id, lun )

```

```

int id, lun;
{
    unsigned char v[ 6 ], buf[ 80 ];
    int i;
    v[ 0 ] = C.TSTRDY;
    v[ 1 ] = ( lun << 5 );
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    dma = 0; /* not DMA mode */
    return ( hdcad( id, v, buf ) );
}

/* Mode select */
modeselect( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    int i;
    v[ 0 ] = C.MODESEL;
    v[ 1 ] = ( lun << 5 ) + 0x10;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    for( i = 0; i < 12; i++ ) {
        buf[ i ] = 0x00;
    }
    buf[ 3 ] = 0x08;
    buf[ 10 ] = ( ( sectsiz == 256 ) ? 0x01 : 0x02 ); /* 256 or 512 byte / sector */
    /*
    buf[ 12 ] = 0x01;
    buf[ 13 ] = 0x06;
    buf[ 14 ] = 0x24;
    buf[ 15 ] = 0x08;
    buf[ 16 ] = 0x08;
    buf[ 17 ] = 0x00;
    buf[ 18 ] = 0x00;
    buf[ 19 ] = 0x00;
    */
    dma = 0; /* not DMA mode */
    if( hdcad( id, v, buf ) ) {
        if( requestsense( id, lun, sensbuf ) == 6 ) {
            /* reset condition */
            hdcad( id, v, buf );
        }
        else {
            printsens( sensbuf );
        }
    }
}

/* read capacity */
readcapacity( id, lun, buf )
int id, lun;
{
    unsigned char *buf;
    {
        unsigned char v[ 10 ], sensbuf[ 256 ];
        v[ 0 ] = C.RDCAP;
        v[ 1 ] = lun << 5;
        v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00; /* logical block address is 0 */
        v[ 6 ] = v[ 7 ] = 0x00; /* PMW bit false */
        v[ 8 ] = 0x00; /* flag and link false */
        v[ 9 ] = 0x00;
        dma = 0; /* not DMA mode */
        if( hdcad( id, v, buf ) ) {
            if( requestsense( id, lun, sensbuf ) == 6 ) {
                /* reset condition */
                hdcad( id, v, buf );
            }
            else {
                printsens( sensbuf );
            }
        }
    }
}

/* Format unit */
format( id, lun, inlv )

```

```

int id, lun, inlv;
{
    unsigned char v[ 6 ], sensbuf[ 256 ];
    v[ 0 ] = C.FMTDRV;
    v[ 1 ] = lun << 5;
    v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00;
    v[ 4 ] = inlv; /* not DMA mode */
    dma = 0; /* Physical format executing. Wait 2 to 5 min. Yn */
    printf( "Physical format executing. Wait 2 to 5 min. Yn\n" );
    if( hdcad( id, v, buf ) ) {
        if( requestsense( id, lun, sensbuf ) == 6 ) {
            /* reset condition */
            hdcad( id, v, sensbuf );
        }
        else {
            printsens( sensbuf );
        }
    }
}

/* Mode select */
modeselectfirst( id, lun )
int id, lun;
{
    unsigned char v[ 6 ], buf[ 128 ], sensbuf[ 256 ];
    int i, cyl, hd, wp, rwc;
    do {
        printf( "Yn Enter the cylinder no. of HDD\n" );
        scanf( "%d", &cyl );
        printf( "Yn Head no. of HDD\n" );
        scanf( "%d", &hd );
        printf( "Yn Enter the 'Write precomp cylinder' of Yn\n" );
        printf( "HDD ( If it's not necessary, enter the Yn )\n" );
        printf( "%s", "No. of Cyl." );
        scanf( "%d", &wp );
        printf( "Yn Enter the 'Reduced write current cylinder' Yn\n" );
        printf( "of HDD ( If it's not necessary, enter Yn )\n" );
        printf( "%s", "the same No. of Cyl." );
        scanf( "%d", &rwc );
        printf( "Yn Are you sure ? (Y/N)\n" );
        scanf( "%s", buf );
        while( !strcmp( buf, "Y" ) ) {
            for( i = 0; i < 127; i++ ) {
                buf[ i ] = 0;
            }
            v[ 0 ] = C.MODESEL;
            v[ 1 ] = ( lun << 5 ) + 0x10;
            v[ 2 ] = v[ 3 ] = v[ 4 ] = v[ 5 ] = 0x00; /* parameter byte no. */
            v[ 4 ] = 56; /* block header */
            buf[ 0 ] = 0x00;
            buf[ 1 ] = 0x00;
            buf[ 2 ] = 0x00;
            buf[ 3 ] = 0x08; /* block descriptor */
            buf[ 4 ] = 0x00;
            buf[ 5 ] = 0x00;
            buf[ 6 ] = 0x00;
            buf[ 7 ] = 0x00;
            buf[ 8 ] = 0x00;
            buf[ 9 ] = 0x00;
            buf[ 10 ] = ( ( sectsiz == 256 ) ? 0x01 : 0x02 ); /* 256 or 512 byte / sector */
            buf[ 11 ] = 0x00;
            buf[ 12 ] = 0x04; /* page code 4 */
            buf[ 13 ] = 0x00; /* parameter byte no. */
            buf[ 14 ] = 0x00; /* No. of Cylinders */
            buf[ 15 ] = cyl > 8;
            buf[ 16 ] = cyl > 8;
            buf[ 17 ] = cyl & 0x00ff; /* No. of heads */
            buf[ 18 ] = 0x00;
            buf[ 19 ] = 0x00; /* write precomp start cylinder */
            buf[ 20 ] = wp > 8;
            buf[ 21 ] = wp > 8;
            buf[ 22 ] = rwc > 8; /* reduced write current start cylinder */
            buf[ 22 ] = rwc > 8;
        }
    }
}

```



```

buf[ 23 ] = rec & 0x00ff;
/* page code 3 */
buf[ 24 ] = 0x03;
buf[ 25 ] = 0x05;
buf[ 26 ] = 0x00;
buf[ 27 ] = 0x00;
buf[ 28 ] = 0x00;
buf[ 29 ] = 0x00;
buf[ 30 ] = 0x00;
buf[ 31 ] = 0x00;
buf[ 32 ] = 0x00;
buf[ 33 ] = 0x00;
buf[ 34 ] = 0x00;
buf[ 35 ] = 0x00;
buf[ 36 ] = 0x00;
buf[ 37 ] = 0x00;
buf[ 38 ] = 0x00;
buf[ 39 ] = 0x00;
buf[ 40 ] = 0x00;
buf[ 41 ] = 0x00;
buf[ 42 ] = 0x00;
buf[ 43 ] = 0x00;
buf[ 44 ] = 0x00;
buf[ 45 ] = 0x00;
buf[ 46 ] = 0x00;
buf[ 47 ] = 0x00;

/* page code 1 */
buf[ 48 ] = 0x01;
buf[ 49 ] = 0x06;
buf[ 50 ] = 0x24;
buf[ 51 ] = 0x08;
buf[ 52 ] = 0x00;
buf[ 53 ] = 0x00;
buf[ 54 ] = 0x00;
buf[ 55 ] = 0x00;

if( hhead( id, v, buf ) ) {
    if( requestsense( id, lun, sensbuf ) == 6 ) {
        hhead( id, v, buf );
    } else {
        printsens( sensbuf );
    }
}

/* Format Unit */
formatfirst( id, lun, inlv );
int id, lun, inlv;
{
    unsigned char v[ 6 ]; sensbuf[ 256 ];
    struct {
        int dfno, i;
        int cyl;
        int head;
        int diff;
    } defect[ 100 ];
    char buf[ 1024 ]; #ptr;
    printf( "Enter the Defect No. on this HDD? " );
    scanf( "%d", &dfno );
    do {
        for( i = 0; i < dfno; i++ ) {
            printf( "Defect %3d: Enter the Cylinder? ", i + 1 );
            scanf( "%d", &(defect[ i ].cyl) );
            printf( "Defect %3d: Enter the Head No. ? " );
            scanf( "%d", &(defect[ i ].hd) );
            printf( "Defect %3d: Enter the position from Index? " );
            scanf( "%d", &(defect[ i ].diff) );
        }
        putchar( '\n' );
    }
    for( i = 0; i < dfno; i++ ) {
        printf( "Defect %4d: Cylinder %4d Head%3d %3dByes\n",
            i + 1, defect[ i ].cyl, defect[ i ].hd, defect[ i ].diff );
        printf( "Where you sure? (Y/N) " );
        scanf( "%s", buf );
        while( #buf != 'y' && #buf != 'Y' );
    }
}

```

```

v[ 0 ] = C.FMTDRV;
v[ 1 ] = ( lun << 5 ) + 0x1e;
v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
v[ 4 ] = inlv;

buf[ 0 ] = 0x00; /* create p-list */
buf[ 1 ] = 0x01;
buf[ 2 ] = ( dfno * 8 ) >> 8;
buf[ 3 ] = ( dfno * 8 ) & 0x00ff;

ptr = &( buf[ 4 ] );
for( i = 0; i < dfno; i++ ) {
    #ptr++ = defect[ i ].cyl >> 8;
    #ptr++ = defect[ i ].hd & 0x00ff;
    #ptr++ = defect[ i ].hd;
    #ptr++ = defect[ i ].diff >> 8;
    #ptr++ = defect[ i ].diff & 0x00ff;
    #ptr++ = 0;
    #ptr++ = 16;
    #ptr++ = 0;
}
dma = 0; /* not DMA mode */
printf( "Physical format executing. wait 2 to 5 min. Yn" );
if( hhead( id, v, buf ) ) {
    if( requestsense( id, lun, sensbuf ) == 6 ) {
        /* reset condition */
        hhead( id, v, buf );
    } else {
        printsens( sensbuf );
    }
}

/* Write 1 Sector */
write( id, lun, lsn, buf );
int id, lun; /* logical sector number */
long lsn; /* logical sector number */
unsigned char #buf; /* data buffer */
{
    unsigned char v[ 6 ];
    unsigned char sensbuf[ 256 ];

    if( scsiz == 256 ) {
        lsn = 2L;
    }
    v[ 0 ] = C.WTSEC;
    v[ 4 ] = ( scsiz == 256 ) ? 2 : 1;
    v[ 5 ] = 0x00;
    v[ 1 ] = ( lun << 5 ) + ( lsn >> 16 ) & 0x00ff;
    v[ 2 ] = ( lsn >> 8 ) & 0x00ff;
    v[ 3 ] = lsn & 0x00ff;
    dma = 0; /* write 256 bytes */
    scsimize( &( hhead( id, lun, sensbuf ) == 6 ) );
    if( hhead( id, lun, sensbuf ) == 6 ) {
        /* reset condition */
        hhead( id, v, buf );
    } else {
        printsens( sensbuf );
    }
}

requestsense( id, lun, buf );
int id, lun;
unsigned char #buf;
{
    /* check error, do request sense. */
    unsigned char v[ 6 ];
    v[ 0 ] = C.RECSEN;
    v[ 1 ] = lun << 5; /* set Lun */
    v[ 2 ] = v[ 3 ] = v[ 5 ] = 0x00;
    v[ 4 ] = 255;
    dma = 0x00;
    hhead( id, v, buf );
    return( buf[ 2 ] );
}

```

```

)
/* disconnect */
treg = 0;
if (treg == 0x01) {
    /* parity error */
    printf("parity error occurred.\n");
    exit(99);
} else if (treg == 0x02) {
    /* check condition */
    return(1);
} else {
    return(treg);
}

/* manual Data transfer routine */
manif(buf)
unsigned char *buf;
{
    int treg; /* temporary register */
    int tphase; /* initial phase storage */
    while( ( treg = inp( P_SNS ) ) & REQ ) {
        /* wait until REQ = true */
        tphase = treg & ( MSG + CD + 10 );
        do {
            tfrsub(buf++);
            while( ( treg = inp( P_SNS ) ) & REQ ) {
                /* wait until REQ = true */
            }
            treg = ( MSG + CD + 10 );
        } while( treg == tphase ); /* repeat until phase change */
        return( 0 );
    }

    /* sub routine: 1 byte transfer */
    tfrsub(buf)
    unsigned char *buf;
    {
        int treg; /* temp. reg. */
        treg = CD;
        while( treg ); /* wait until command phase */
        manif(sc); /* output command data */
        /* data transfer phase */
        printf("Data Transfer.\n");
        while( ( treg = inp( P_SNS ) ) & REQ ) {
            /* wait while REQ = false */
        }
        if( ( treg & ( MSG + CD + 10 ) ) != 0 ) { /* status phase? (MSG=f, CD, 10=t) */
            /* no. data phase. */
            manif(buf);
        }
        /* status phase */
        printf("Status.\n");
        do {
            while( ( treg = inp( P_SNS ) ) & REQ ) {
                /* wait while REQ = false */
            }
            while( ( treg & ( MSG + CD + 10 ) ) != 0 ); /* wait until MSG=f CD, 10=t */
            tfrsub(&tbuf); /* get Status byte */
        }
        /* message phase */
        printf("Message.\n");
        do {
            while( ( treg = inp( P_SNS ) ) & REQ ) {
                /* wait until REQ = true */
            }
            treg = ( MSG + CD );
            while( treg != MSG );
            tfrsub(&msgin); /* get message byte */
        }
        /* bus free phase */
        printf("Bus Free.\n");
        while( ( inp( P_SNS ) & BSV ) ) {
            /* wait until BSV = false */
        }
    }
}

```

```

)
/* disconnect */
treg = 0;
if (treg == 0x01) {
    /* parity error */
    printf("parity error occurred.\n");
    exit(99);
} else if (treg == 0x02) {
    /* check condition */
    return(1);
} else {
    return(treg);
}

/* manual Data transfer routine */
manif(buf)
unsigned char *buf;
{
    int treg; /* temporary register */
    int tphase; /* initial phase storage */
    while( ( treg = inp( P_SNS ) ) & REQ ) {
        /* wait until REQ = true */
        tphase = treg & ( MSG + CD + 10 );
        do {
            tfrsub(buf++);
            while( ( treg = inp( P_SNS ) ) & REQ ) {
                /* wait until REQ = true */
            }
            treg = ( MSG + CD + 10 );
        } while( treg == tphase ); /* repeat until phase change */
        return( 0 );
    }

    /* sub routine: 1 byte transfer */
    tfrsub(buf)
    unsigned char *buf;
    {
        int treg; /* temp. reg. */
        outp( P_DATA, 0xff ); /* clear data line */
        while( ( treg = inp( P_SNS ) ) & REQ ) {
            /* wait while REQ = false */
        }
        if( ( treg & 10 ) == 0 ) {
            outp( P_CNT, 105+DIR+ATN );
            /* 1/2 phase */
            tbuf = inp( P_DATA );
            outp( P_CNT, ATN );
        } else {
            /* 1/2 OUT */
            outp( P_DATA, ~tbuf );
        }
        outp( P_CNT, ATN + ACK ); /* set ACK bit */
        while( ( inp( P_SNS ) & REQ ) ) {
            /* wait until REQ = false */
        }
        outp( P_CNT, ATN ); /* reset ACK bit */
        outp( P_DATA, 0xff ); /* clear data line */
    }
}

dump(buf)
unsigned char *buf;
{
    int i, j;
    unsigned d;
    printf(" 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF\n");
    for( i = 0; i < 256; i += 16 ) {
        printf(" %02X: ", i );
        for( j = 0; j < 16; j++) {
            printf(" %02X ", buf[ i + j ] );
        }
    }
}

```



リスト12> SFEHEC (つづき)

```
for(i=0; i<16; i++){
    d = buf[i];
    if((d>= ' ') && (d< 0x7f)){
        putchar(d);
    }else{
        putchar(' ');
    }
    putchar(' ');
}
```

リスト13> EASYHARD.H

```
/* I/O address */
#define P_DATA 0x378
#define P_SNS 0x379
#define P_CNT 0x37a

/* transfer Mode */
#define MSG 0x80
#define REQ 0x40
#define HSX 0x20
#define CD 0x10
#define IO 0x08

#define IOS 0x80
#define IIR 0x20
#define ISY 0x08
#define ATN 0x04
#define SEL 0x02
#define ACK 0x01

/* Controller command */
#define C_TSTRDY 0x00
#define C_TZBR 0x01
#define C_BEGSN 0x02
#define C_FMRDY 0x04
#define C_REIDSEC 0x08
#define C_WRTSEC 0x0A
#define C_SEEK 0x0B
#define C_MDSSEL 0x15
#define C_MDSSEN 0x1A
#define C_RDGAP 0x25

/* SCSI ID */
#define INITID 7
#define INITBIT 0x80
#define TAGTID 0
#define TAGTBIT 0x01

/* Global Variables */
/* data buffer for system */
char sysbuf[2048];

char stbuf; /* buffer for status byte */
char msgin; /* buffer for message byte */
int dma; /* interrupt status */
int dmaode;
int scsinode;
```

エンジニアのためのスーパーMOOK

# 最新SCSIマニュアル

インターフェース  
編集部編  
2色刷 B5判 192頁  
定価1,650円(税込み)

●標準入出力インターフェースの規格・使い方・設計ノウハウ

SCSIへのやさしい入門から、PC-98/286やMacintoshのSCSI活用事例、それにSCSI規格の詳細解説やSCSI-2最新情報までを幅広く網羅した、わが国初のSCSI技術実用書です。

IFビギナーズ I/F編集部編、2色刷176頁、定価1,648円(税込み)

## OS/2プログラミング

●MS-DOSソフトの移植からマルチタスク・プログラム作成まで

OS/2は、80286/386を有効に使用し、マルチタスク、仮想メモリなどをサポートした次世代パソコンOSです。本書は、OS/2のアーキテクチャから実際のマルチタスク・プログラミングまでを幅広く解説します。

IFビギナーズ 西沢 昭著、224頁、定価1,650円(税込み)

## Z80上級プログラミング

●制御用を中心にした実践アセンブラ技法

高精度四則演算から割込み多重処理まで、制御関係を中心に実務に欠かせないZ80アセンブラ技法を網羅。

IFビギナーズ 岡村周善著、2色刷224頁、定価1,648円(税込み)

## VMEシステム完全マスタ

●68010ボードの設計からOS-9/68Kの移植まで

VMEバス規格の基礎をていねいに解説した後、VMEバス・システムの構築技術を紹介しています。

CQ出版社

〒170 東京都豊島区巣鴨1-14-2

振替 東京0-10665

# SPC(MB89352)の解説

宇野沢成夫

富士通の SCSI プロトコル・コントローラにはいくつかのバリエーションがありますが MB89352 は同期転送はサポートしていないもののドライバ/レシーバを内蔵しており、非常に簡単な回路でパソコンなどの SCSI インターフェースを作成することができます。

MB89352 は 8 バイト FIFO バッファおよびそれをコントロールする転送バイト・カウンタを内蔵しており、簡単な手順で高速なデータ転送が可能になっています。

パッケージは 48 ピン DIP と 48 ピン QFP の 2 種類があります(図 A)。

MB89352 のデータ・シートにも記載されていますが、ここでは SCSI プロトコル・コントローラを略して SPC と呼びます。SPC の場合、イニシエータとターゲット両方に使用できるのですがここではイニシエータとして使う場合に限りて説明します。

## ● MB89352 のレジスタ

MB89352 には 14 バイトのアクセス可能なレジスタがあります(表 A)。

### (1) BDID レジスタ

BDID レジスタは、SCSI ID を書き込みます。読み込み時は書き込まれた ID に対応する 1 ビットがセットされています。

### (2) SCTL レジスタ

SCTL レジスタは SPC の動作を設定するレジスタ

で、通常 BDID レジスタとともに初期化のときに設定します。

ビット 7 の Reset & Disable を“1”にセットすることで SCSI バスと論理的に切り離された状態になり、SPC 内部のリセットを行います。また、CPU 側よりハードウェア・リセットを行われた際“1”にセットされるので SPC を使うときに BDID レジスタへ SCSI ID をセットした後、“0”にセットします。

ビット 6 の Control Reset は SPC のデータ転送制御回路に対するリセットで、“1”の間りセットが実行されますが、SCSI バスと結合中でも結合関係は保持されます。

ビット 5 を“1”にした場合 SPC は診断モードと呼ばれるモードになり SCSI とは論理的に切り離され SDGC レジスタにより擬似的に SCSI 上の動作を実行することができます。

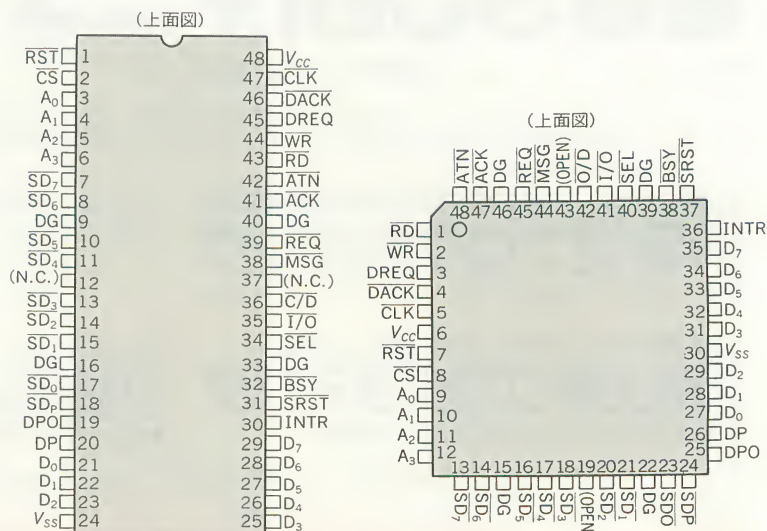
ビット 4 はアービトレーション、ビット 3 はパリティ・チェックを実行するかどうか指定するビットです。

ビット 2 はセクションに応答するかどうかを指定するビットなのですが、イニシエータのみの場合このビットを“0”にセットします。

ビット 1 はリセクションに応答するかどうかを指定するビットで、ディスクコネクをドライバがサポートする場合にはこのビットを“1”にします。

ビット 0 は SPC からインタラプトを許可するかどうか

〈図 A〉  
MB89352 のピン配置



(a) DIP 端子配列図

(b) フラット・パッケージ端子配列図



うかを指定するビットで、“1”にした場合はインタラプトを許可します。

### (3) SCMD レジスタ

SPC の場合、SCSI バスへなにかの動作をするためにはコマンドをこの SCMD レジスタへ発行するという形態をとっています。ビット 7, 6, 5 へはコマンドを書き込みます。コマンドは表 B のような種類があり、このレジスタへ書き込まれた時点で実行されます。

ビット 4 は SCTL レジスタが“0”のときにこのビットを“1”にすることで SCSI バスの RST 信号をアサートすることができます。

ビット 3 を“1”にして Set ACK/REQ と Reset

ACK/REQ コマンドを用いてマニュアル転送を行う場合は SPC 内の 8 バイト FIFO レジスタの内容は保

〈表 B〉 SCMD レジスタ

ビット 7	ビット 6	ビット 5	コマンド
0	0	0	Bus Release
0	0	1	Select
0	1	0	Reset ATN
0	1	1	Set ATN
1	0	0	Transfer
1	0	1	Transfer Pause
1	1	0	Reset ACK/REQ
1	1	1	Set ACK/REQ

〈表 A〉 内部レジスタのアドレスおよびビット割り付け

アドレス	名 称 (略称)	R/ W	レ ジ ス タ ・ ビ ッ ト								パリティ				
			7	6	5	4	3	2	1	0					
0	Bus Device ID (BDID)	R	# 7	# 6	# 5	# 4	# 3	# 2	# 1	# 0	0				
		W	—						ID4	ID2	ID1	—			
1	Spc Control (SCTL)	R	Reset & Disable	Control Reset	Diag Mode	ARBIT Enable	Parity Enable	Select Enable	ReSelect Enable	INT Enable	P				
		W	—				—	—	—	—	—	—			
2	Comand (SCMD)	R	Command Code			RST	Intercept	Transfer Modifier		Term	P				
		W	—			Out	Xfer	PRG Xfer	0	Mode	—				
4	Interrupt Sense (INTS)	R	Selected	ReSe- lected	Dis- connected	Command Complete	Service Required	Time Out	SPC Hard Error	Reset Condition	P				
		W	Reset Interrupt									—			
5	Phase Sense (PSNS)	R	REQ	ACK	ATN	SEL	BSY	MSG	C/D	I/O	P				
	SPC Diag. Control (SDGC)	W	Diag REQ	Diag ACK	Xfer Enable	—	Diag BSY	Diag MSG	Diag C/D	Diag I/O	—				
6	SPC Status (SSTS)	R	Connected INIT   TARG		SPC BSY	Xfer in Progress	SCSI RST	TC=0	DREG FULL	Status EMPTY	P				
7	SPC Error Status (SERR)	R	Data Error		Xfer	0	TC	0	Short	0	P				
		W	SCSI	SPC	Out	0	P-Error	0	Period	0	—				
8	Phase Control (PCTL)	R	Bus Free Interrupt Enale	0				Transfer Phase		P					
		W	—				MSG Out	C/D Out	I/O Out	—					
9	Modified Byte Counter (MBC)	R	0				MBC				P				
A	Data Register (DREG)	R	Internal Data Register(8 バイト FIFO)								P				
		W	7	6	5	4	3	2	1	0	—				
B	Temporary Register (TEMP)	R	7	6	Temporary Data (Input : From SCSI)				5	4	3	2	1	0	P
		W	7	6	Temporary Data (Output : To SCSI)				5	4	3	2	1	0	P
C	Transfer Counter High (TCH)	R	Transfer Counter High (MSB)								P				
		W	23	22	21	20	19	18	17	16	—				
D	Transfer Counter Mid (TCM)	R	Transfer Counter Middle(2nd バイト)							—		P			
		W	15	14	13	12	11	10	9	8	—				
E	Transfer Counter Low (TCL)	R	Transfer Counter Low (LSB)								—		P		
		W	7	6	5	4	3	2	1	0	—				

- (注) 1. 書き込み動作において一が記入されているビットは、1, 0 のどちらを書き込んでもよいことを示します。  
2. 読み出し専用レジスタへの書き込み動作は無視されます。  
3. 読み出し時において 0 と示されているビットは必ず 0 が読み出されます。



存されます。これは、データ転送フェーズの最中にメッセージ・フェーズへのフェーズの変更による Service Require 割り込みの処理などに用いられます。

ビット 2 は Transfer コマンドを実行するときに DMAC を使用するための DREQ を出すかどうかを指定するビットで“0”のときに DREQ を出します。

ビット 0 は Transfer コマンド実行中に転送バイト・カウンタがゼロになった場合の処理を指定するビットで、“0”を指定した場合はバイト・カウンタがゼロになった時点で Transfer コマンドを終了します。

“1”を指定した場合、アウトプット動作中であれば OOH のデータをフェーズの変更があるまで転送しつづけます。また、インプット動作中であれば受領データをフェーズ変更があるまで無視します。これは Padding 転送と呼び、SPC 内部で実行されるので DMA 転送時の DREQ やプログラム転送時の DREQ レジスタ・アクセス要求の INTR 信号は出されません。

#### (4) INTS レジスタ

割り込み原因の表示およびリセットを行うためのレジスタです。SPC からの割り込みはビット 0 の Reset Condition を除いて SCTL レジスタのビット 0 によりマスク可能です。

割り込みのリセットはこのレジスタに表示される割り込み原因のビット位置に“1”を書き込むことにより行います。ビット 7 は SCSI バス上の他のイニシエータよりセレクション・フェーズによりセレクトされたことを示します。この場合 SPC は Bus Release コマンドを発行するリセット・コンディションを検出するまでターゲット・モードになります。

ビット 6 は SCSI バス上のほかのターゲットよりセレクション・フェーズによりセレクトされたことを示します。

ビット 5 は PCTL レジスタのビット 7 が“1”のとき SCSI バスが、バス・フリーへ移行したことを示します。

ビット 4 は Select または Transfer コマンドの動作が終了したことを示します。

ビット 3 は Transfer コマンド動作中に転送フェーズが PCTL レジスタのビット 2, 1, 0 に指定されているものと違うフェーズの場合を示します。

ビット 2 は Select コマンドによりセレクション・フェーズを実行したが指定された監視時間を経過してもターゲットから応答がなかったことを示します。

ビット 1 は SPC が SERR レジスタのビット 3, 1 に表示されるエラーがあることを示します。

ビット 0 は SCSI バス上でリセット・コンディションを検出したことを示します。この割り込みを検出した場合 SST レジスタのビット 3 が“0”であることを

確認した上で、この割り込みをリセットします。

#### (5) PSNS レジスタ

SCTL レジスタのビット 5 が“0”のときは SCSI バス上の信号線の状態を示し、“1”のときは SPC から送出する信号の状態を示します。

#### (6) SDGC レジスタ

SCTL レジスタのビット 5 が“1”のとき(診断モード)SCSI バス上の信号線を擬似的に動作させるために使用します。

#### (7) SSTS レジスタ

SPC の内部のステータスを表示するレジスタです。ビット 7, 6 は SPC と SCSI の結合状態を示します。

ビット 7	ビット 6	状 態
0	0	非結合中
1	0	イニシエータとして結合中
0	1	ターゲットとし結合中
1	1	未定義

ビット 5 はコマンドの実行中または実行待ちの状態であることを示します。

ビット 4 はハード転送中または SCSI バス上で転送フェーズが要求されていることを示します。

ビット 3 は SCSI バス上の RST 信号の状態を示します。

ビット 2 は転送バイト・カウンタがゼロであることを示します。

ビット 1, 0 は FIFO バッファの状態を示します。

プログラム転送時はこのビットの値を見ることにより、データ・バッファへのアクセス・タイミングを知ることができます。

ビット 1	ビット 0	状 態
0	0	1~7 バイトのデータを保持している
0	1	空の状態
1	0	8 バイトのデータを保持していて空なし
0	1	未定義

#### (8) SERR レジスタ

SPC 内部で検出されたエラーの詳細を表示するレジスタです。

ビット 7, 6 は SCSI の転送フェーズを実行中にパリティ・エラーが検出されたことを示します。

ビット 3 は転送バイト・カウンタの減算動作でパリティ・エラーが検出されたことを示します。

ビット 1 は SCSI バス上の REQ/ACK 信号が SPC の追従範囲を超えた(速すぎる)周期で入力されたことを示します。

#### (9) PCTL レジスタ

ビット 7 を“1”に設定したとき、SCSI バス上でバス・フリー・フェーズを検出すると、Disconnect 割り込みが発生します。Select コマンド発行時および Dis

connect 割り込みをリセットするときは必ずこのビットを“0”にセットします。

なお、Select コマンドを実行する場合はコマンドの発行に先だって、OOH を設定するとセクション・フェーズ、O1H を設定するとリセクション・フェーズを実行します。また、SCSI バスと結合中の場合は実行予定のフェーズをビット 2, 1, 0 へ指定します。

#### (10) MBC レジスタ

SPC の FIFO バッファと MPU またはメモリとのデータ転送数を制御するカウンタです。

#### (11) DREG レジスタ

SPC の FIFO データ・バッファで Transfer コマンド実行時このレジスタを介してデータを転送します。

#### (12) TEMP レジスタ

ハード転送以外の場合に SCSI データ・バスを制御するための送信/受信の独立したレジスタです。

#### (13) 転送バイト・カウンタ・レジスタ (TCH, TCM, TCL)

3 バイト構成の減算カウンタでハード転送動作実行時に 1 バイトのデータを転送するごとに 1 バイト減らされ、転送すべき残りバイト数を表します。

また、Select コマンドの実行時には応答待ち時間監視用のタイマとして動作します。

### ● MB89352 のコマンド

MB89352 の SCSI バスへの動作はコマンドの実行 (SCMD レジスタへの書き込み) により行われます。

#### (1) Bus Release コマンド

ターゲットとして動作中のときはバス・フリー・フェーズへ移行しますが、イニシエータではバス・フリー待ちの Select コマンドを解除する場合に用います。

#### (2) Select コマンド

このコマンドを受け取るとバス・フリー・フェーズを検出後 SCTL レジスタのビット 4 が“1”であればアービトラーションを行いバス使用权を獲得した場合 PCTL レジスタのビット 0 が“0”の場合セクション、“1”の場合リセクション・フェーズを実行します。SCTL レジスタのビット 4 が“0”であればバス・フリー・フェーズを検出後にセクション・フェーズを実行します。

このコマンドを発行する前に TCH, TCM レジスタへセクション・フェーズにおける応答時間 ( $T_{st}$ ) を TCL レジスタへ、バス・フリー検出後アービトラーションおよびセクション・フェーズへ移行するまでの時間 ( $T_{wait}$ )、TEMP レジスタへターゲットおよびイニシエータのデバイス ID を設定します。

また、セクション・フェーズを実行する際に SCSI バスへ ATN 信号を送出する必要がある場合は、前もって Set ATN コマンドを実行しておきます。

▶ TCH, TCM へ書き込む値  $N$  を算出する式

$$N \neq 0 \text{ のとき } T_{st} = (N \times 256 + 15) \times T_{clt} \times 2$$

$$N = 0 \text{ のとき } T_{st} = \infty$$

( $T_{clt}$  は SPC の CLK 端子に供給するクロックの周期)

#### ▶ TCL へ書き込む値の推奨値

$T_{clt}$ (ns)	TCL	$T_{wait}$ [平均 (ns)]
125~180	O4H	1250~1980
140~200	O3H	1260~2000

#### (3) Set ATN および Reset ATN

SCSI バス上の ATN 信号をセット/リセットするコマンドです。

#### (4) Transfer コマンド

SCSI の情報転送フェーズを実行するコマンドで、これによって行われる転送をハード転送と呼びます。ハード転送はさらに SCMD レジスタのビット 2 によりプログラム転送モードと DMA 転送モードに分かれます。このコマンドでは SCSI バス上の REQ/ACK のハンドシェイクなどは SPC が制御するので、コマンド発行前に転送バイト・カウンタ (TCH, TCM, TCL), PCTL レジスタの実行フェーズを示すビット・パターンの初期設定が必要になります。イニシエータとして動作しているときは、REQ 信号を受領したときに設定したフェーズが一致した場合に転送を開始します。転送終了は Disconnect 割り込みの発生時、Padding 転送モードでない場合は転送バイト・カウンタがゼロになった時点、Padding 転送時はフェーズが不一致した時点で終了します。

プログラム転送モードでは DREG レジスタを介してデータ転送を行います。DREG レジスタの状態を知る方法としては 2 種類あります。ひとつは SSTS レジスタを読み出すことによりポーリングを行う方法、もうひとつは SDGC レジスタのビット 5 を“1”にすることによって DREG レジスタが FULL か EMPTY かを INTR による割り込みによって知る方法です。

DMA 転送の場合は DREQ 信号とその応答による DACK によって行いますが、DREQ 信号は転送するデータが DREG レジスタ内に存在する間は常にアサートされた状態になるので、エッジ・トリガを必要とする DMAC を使用する場合は若干の回路修正が必要になります。

#### (5) Set ACK/REQ および Reset ACK/REQ コマンド

SCSI バス上の REQ/ACK 信号線のハンド・シェイクをマニュアルで転送する場合のコマンドです。

この場合もあらかじめ PCTL レジスタへ実行フェーズを示すビット・パターンをあらかじめ設定しておかなければなりません。データは TEMP レジスタを介して行われます。



## 第7章

# SCSI2 の概要—拡張と変更を中心に

もう一部では SCSI3 の話も。乗り遅れるな！

菅谷誠一

1986 年に ANSI で SCSI (Small Computer System Interface, ANSI 規格番号: X3.131-1986) の規格化が完了して以来, 中小型システム分野を中心に SCSI の適用が急速に進展し, これらに接続される小型ハードディスク装置, カートリッジ磁気テープ装置, 光ディスク装置などの入出力装置の標準 I/O インターフェースとして SCSI は完全に定着しています。

ANSI では SCSI の規格がまとまった直後から, さらに高機能化および高性能化を図るための検討を行ってきました。この拡張仕様は SCSI2 と呼ばれ, 各種システムからの高性能化の要求に応えるとともに, 適用可能分野の拡大をねらったものです。図 1 に SCSI の標準化の推移を示します。

SCSI2 は, まだ正式に ANSI 規格になったものではありませんが, 1990 年 3 月に ANSI の作業部会 (X3T9.2) から最終規格案 (文書番号: X3T9.2/86-109 Revision 10c) が発行され, 仕様のほとんどが明確になっています。また, すでにこの仕様を適用したハードディスク装置などが登場してきています。今後, 正式に規格化が完了するまでの間に, 仕様の追加などが行われる可能性はありますが, ここでは現在明らかになっている SCSI2 の仕様に基づいて従来の SCSI 仕様との相違および拡張された機能について, ハード

ディスク装置を中心に解説します。

なお, 区別を明確にするために, 従来の SCSI 仕様を SCSI1 と呼ぶことにします。

## SCSI2 の概要

SCSI2 の開発のねらいは SCSI1 との継続性 (下方互換性) を維持しながらの機能拡張および高性能化であるといえます。それらの主要項目の概要はつぎのとおりです。

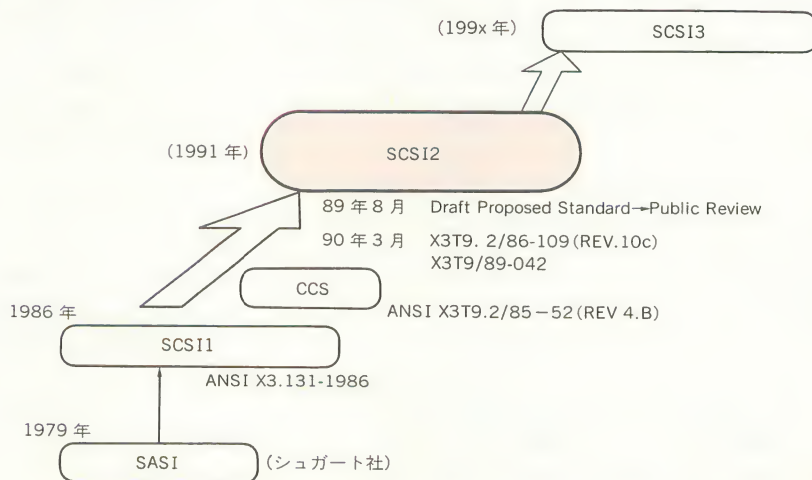
### ● SCSI1 との互換性

SCSI1 では表 1 に示したように, オプションまたは拡張機構として規定されている機能の組み込みの度合いによってレベル 0~2 の機能レベルが定義されていました。また, SCSI1 のベースとなった SASI との互換性が保たれていました。

SCSI2 では SCSI1 でオプションとなっていた, データ・バス・パリティ, アービトレーション・フェーズ, アテンション・コンディションが必須の機能になったため, データ・バス・パリティを使用しているレベル 2 の SCSI1 適用環境との接続互換性のみが確保されています。

したがって SCSI2 適用の装置は, この条件を満た

〈図 1〉 SCSI 標準化の推移





している SCSI1 の装置と同一のバス上に共存させることができます。SASI との整合性は発展的に放棄されていますが、SCSI1 の標準化完了後に製品化されたほとんどの SCSI 装置はレベル 2 の機能を満たしているので、実質的には SCSI1 との互換性が確保されているといえます。

### ● サポート・デバイスの拡充

表 2 に示すように、あらたに 4 種のデバイス・タイプが追加され、各々に標準のコマンド・セットが定義されています。

また、SCSI1 でリード・オンリ・デバイスと呼ばれていたデバイス・タイプは CD-ROM デバイスと呼称が変わり、装置の性格付けを明確にしました。

### ● 高性能化

データ転送の高速化を図るためにふたつの手法が導入されました。

ひとつはインターフェース・ケーブルを追加することでデータ・バスを 2 バイト幅または 4 バイト幅に拡張して高速データ転送を可能にするもので、WIDE SCSI と呼ばれます。

もうひとつの手段は同期モードのデータ転送フェーズのタイミング規約を高速化し、1 バイト幅のデータ・バスで最大 10M バイト/s までのデータ転送を可能とするもので、FAST SCSI と呼ばれます。

これにより、SCSI1 では 4M~5M バイト/s(同期モ

ード)が上限であったデータ転送性能が 1 バイト・バスで最大 10M バイト/s まで、4 バイト・バスで最大 40M バイト/s まで仕様上可能になりました。

### ● 高機能化

まず最初にコマンド・キューイングの機能が追加されたことがあげられます。この機能を使用することでひとつのコマンドの実行が終了するのを待たずに同一のイニシエータから同一のロジカル・ユニットに対して複数個(最大 256 個)のコマンドを発行することができます。

ターゲットは受け取った複数のコマンドを待ち行列(キュー)に入れ、ロジカル・ユニット上でそれらのコマンド列をもっとも効率よく高速に実行するようにコマンドの実行順序を変更することもできます。

また、キャッシュ・メモリを備えて実効入出力性能の向上を図る装置に対してイニシエータからそのキャッシュ機構の動作を制御するためのコマンド・セットやパラメータなどの定義が追加になっています。

これらのふたつの追加機能はシステム内での入出力動作の性能向上を図ることをねらっています。

その他の主要な追加機能としてはロジカル・ユニット上で非同期に発生した事象(たとえば周辺装置の電源が単独で投入され装置がレディ状態になったときなど)をターゲットからイニシエータに通知するための AEN(Asynchronous Event Notification)や入出力

〈表 1〉  
SCSI1 の  
機能レベル

機能項目	機能レベル			ターゲットとしての必要条件		
	レベル 0	レベル 1	レベル 2	レベル 0	レベル 1	レベル 2
アービトレーション・フェーズ		○	○			
COMMAND COMPLETE メッセージ	○	○	○	○	○	○
MESSAGE REJECT メッセージ			○	△	△	○
IDENTIFY メッセージ			○			○
DISCONNECT メッセージ			○			
SAVE DATA POINTER メッセージ			○			
すべての M(サポート必須)レベルのコマンド				○	○	○
すべての E(拡張機能)レベルのコマンド						○
GOOD ステータス	○	○	○	○	○	○
CHECK CONDITION ステータス	○	○	○	○	○	○

○：サポートする必要がある

△：COMMAND COMPLETE 以外のメッセージがあるときにはサポートする必要がある

〈表 2〉  
デバイス・タイプ

デバイス・タイプ・コード(16進)		デバイス・タイプ
SCSI1 の範囲	00h	ダイレクト・アクセス・デバイス
	01h	シーケンシャル・アクセス・デバイス
	02h	プリンタ・デバイス
	03h	プロセッサ・デバイス
	04h	単一書き込み(Write-once)デバイス
	05h	CD-ROM デバイス(SCSI1：リード・オンリ・デバイス)
SCSI2 の範囲	1Fh	未定義のデバイス・タイプ
	06h	スキャナ・デバイス
	07h	光メモリ(Optical Memory)・デバイス
	08h	メディア・チェンジャ(Media Changer)・デバイス
	09h	通信(Communications)デバイス

デバイス・タイプ・コードはスタンダード INQUIRY データ(図 21)に表示される値

動作でエラーが発生したときなどにそのコマンドを発行したイニシエータがエラー情報の収集やエラーの回復処理を実行する間、他のイニシエータからのそのロジカル・ユニットへのアクセスを一時的に排除する(排他制御)手順である **ECA(Extended Contingent Allegiance)** があります。

### ● 互換性確保の強化

SCSI1 のときには多くの機能がオプションあるいは拡張仕様として規定され装置への組み込みが義務づけられていなかったり、仕様の記述に曖昧な点がありました。このため、メーカー間で実装置への組み込み仕様に差異が生じ、互換性に問題を残しました。

そこで ANSI の作業部会ではもっとも需要の大きいダイレクト・アクセス・デバイスについてコマンド体系などの論理仕様の標準形を定めた **CCS(Common Command Set)** 仕様を 1986 年に発表し、装置間の互換性の確保を図りました。CCS 仕様は ANSI の規格にはなっていませんが、その後のハードディスク装置のほとんどはこの仕様に基づいたものとなっています。また、その他のデバイス・タイプについても CCS の考え方が組み込み仕様のモデルとなってきました。

SCSI2 ではこの CCS の思想をすべてのデバイス・タイプについて適用し、組み込み仕様の必要条件を明確にしています。また、バス・フェーズの動作規定、コマンド仕様、例外条件の処理などの記述を明確にするとともにエラー・リカバリ情報の標準化を図るためにセンス・データ上に表示されるエラー原因を表すコード(ASC: Additional Sense Code, ASCQ: Additional Sense Code Qualifier)をすべてのデバイス・タイプについて体系化しています。

\*

\*

以下では SCSI2 で変更あるいは追加になった機能を中心に詳細を解説しますが、基本的には SCSI1 の発展形なので SCSI1 の知識・経験を生かせば、容易に SCSI2 への移行ができると思われます。

## 物理仕様

信号伝送系には SCSI1 と同様に **シングル・エンド型バス** と **ディファレンシャル型バス** が規定されています。電気的条件やタイミング条件、最大ケーブル長も基本的に同一です。

### ● B ケーブル

インターフェース・ケーブルは 50 信号線の A ケーブルと 68 信号線の B ケーブルとから構成されます。基本の A ケーブルは SCSI1 と同一の構成と機能を持っています。B ケーブルはオプションであり、**WIDE SCSI** でデータ・バス幅を 2 バイトまたは 4 バイトに拡張するときだけに必要となるもので、最大 3 バイトま

での拡張データ・バスと B ケーブル専用のデータ転送制御信号である REQB, ACKB, およびターミネータ用電源などが定義されています。

現在の仕様では B ケーブルは WIDE 転送専用データ・バスとしての機能しかなく、しかも WIDE SCSI を組み込むためにはふたつのインターフェース・コネクタが必要になるので小型の機器に適した仕様とはいえません。

ANSI では、1 本のケーブル(68 ピン: 2 バイト転送, 110 ピン: 4 バイト転送)でデータ・バスを拡張し、さらに追加されたデータ・バス・ビットを SCSI ID の拡張に利用してバス上に接続できるデバイスの数を増加させる改善案を検討していますが、SCSI2 の最終規格に盛り込まれるかどうかは不明です。

### ● ケーブル特性

信号伝送系の構成が SCSI1 と同一であり、ケーブルに要求される特性条件も基本的に SCSI1 と変わりはありません。

しかし、**FAST SCSI** を用いる場合には厳しいタイミング条件が要求されるため、以下の電気的条件を満たすシールド型ケーブルが必要であるとされています。

- ・特性インピーダンス: 90~132Ω
- ・信号減衰特性: 5MHz で 0.095dB/m 以下
- ・信号線間の伝搬遅延時間差: 0.20ns/m 以下
- ・直流抵抗: 20°C で 0.230Ω/m 以下

### ● コネクタ

インターフェース・コネクタはつぎのように体系化されます。1.27mm ピッチが追加され、装置の小型化への対応が図られています。

#### ▶ A ケーブル: 50 コンタクト

- ・ノン・シールド型: (1) 1.27mm ピッチ高密度型 (2) 2.54mm ピッチ低密度型
- ・シールド型: (3) 1.27mm ピッチ高密度型 (4) 2.16mm ピッチ・リボン型

#### ▶ B ケーブル: 68 コンタクト

- ・ノン・シールド型: (5) 1.27mm ピッチ高密度型
- ・シールド型: (6) 1.27mm ピッチ高密度型

図 2~図 5 に各コネクタの外観を示します。

このうち A ケーブル用の低密度型ノン・シールド・コネクタ(図 3)とリボン型シールド・コネクタ(図 5)は SCSI1 と物理的互換性を持っています。なお、SCSI2 で新規に規定された高密度型コネクタは装置側がソケット(レセプタクル)で、ケーブル側がプラグ(ピン・コンタクト)になっているので注意してください。

各コネクタ上でのシングル・エンド型バスとディファレンシャル型バスの信号割り付けを表 3~表 6 に示します。

A ケーブルは SCSI1 互換の信号割り付けになって



いますが、TERMPWR 信号(ターミネータ用電源)の両サイドのピンは“リザーブ”(従来はグラウンド)になりました。これらのピン(将来の規格で使用される可能性がある)はターミネータ・モジュールあるいはバス端に接続される装置内でグラウンドに接続します。それ以外の装置ではオープンにすることが原則ですが、グラウンドに接続することも許容されています。

### ● 電気的条件

ドライバ/レシーバおよびターミネータの電気的条件に関する規定は以下に述べる事項をのぞき、原則的に SCSI1 と同一です。表 7 に電気的条件の一覧と規定の相違を示します。

#### ▶ シングル・エンド型バスのターミネータ

シングル・エンド型バスは元々ノイズ・マージンが厳しく、システムを構成するときのネックになっていました。信号伝送特性を改善するために図 6 に示すターミネータ回路が新たに追加されました。SCSI1 と同一の抵抗分圧型 (220-330Ω) ターミネータも併記されており、必要に応じてどちらかを選択して使用することができます。

#### ▶ ターミネータ用電源の供給

SCSI1 では TERMPWR の使用はオプションとなっていました。SCSI2 ではターミネータには必ず TERMPWR (B ケーブルは TERMPWRB) から電源を供給することとされています。そして通常イニシエータとして動作する装置がその電源を供給する必要があります。ターゲットとしてだけ動作する装置または COPY コマンドや AEN のときにだけ一時的にイニシエータとして動作する装置は、電源を供給できるようにしても構いませんが、その義務はありません。

### ● SCSI バス信号

SCSI1 との相違はつぎのとおりです。

#### ▶ データ・バス・パリティ

パリティ・ビット(奇数パリティ)を使用することが必須条件になりました。

#### ▶ SEL 信号

SEL 信号の駆動方法は OR-tied 型とすることになりました。しかし、この規定は将来の規格拡張に備えたもので Non-OR-tied で駆動している従来の装置と混在接続しても問題はありありません。

#### ▶ B ケーブル信号

WIDE SCSI 用の 3 バイトの拡張データ・バスと転送制御用の REQB、ACKB 信号が追加されました。B ケーブル上のデータ転送のタイミング(ハンドシェイク)は専用の REQB/ACKB によって制御されます。データ・バス・ビットと各バイトごとのパリティ・ビットとの対応はつぎのとおりです。

・ DB<sub>15</sub>~DB<sub>8</sub> : DB<sub>p1</sub>

・ DB<sub>23</sub>~DB<sub>16</sub> : DB<sub>p2</sub>

・ DB<sub>31</sub>~DB<sub>24</sub> : DB<sub>p3</sub>

### ● SCSI バス・タイミング

タイミング規約の一覧を表 8 に示します。FAST SCSI のタイミングが規定された他、今までは各装置の組み込み仕様に依存していた電源投入後またはリセット・コンディション解除後に特定のコマンドに応答できるようになるまでの時間規定(Power-On to Selection Time, Reset to Selection Time: ただし、推奨値として)などが追加されました。

また、Arbitration Delay の規定値が変更になっていますが、これは従来の規定値 (2.2μs) が最大ケーブル長 (25m) で両端の装置がアービトレーション・フェーズに競合したときに矛盾(実動作ではほとんど問題とならない)していたために修正したものです。

## バス・プロトコル

バス・フェーズやバス・コンディションの動作規定については、アービトレーション・フェーズ、リセレクト・フェーズおよびアテンション・コンディションがオプションではなくなりました。

また、以下に述べる変更や追加がありますが、プロトコル規定とタイミング条件は SCSI1 と同等で、同一バスへの混在接続を可能としています。

### ● セレクション・フェーズ

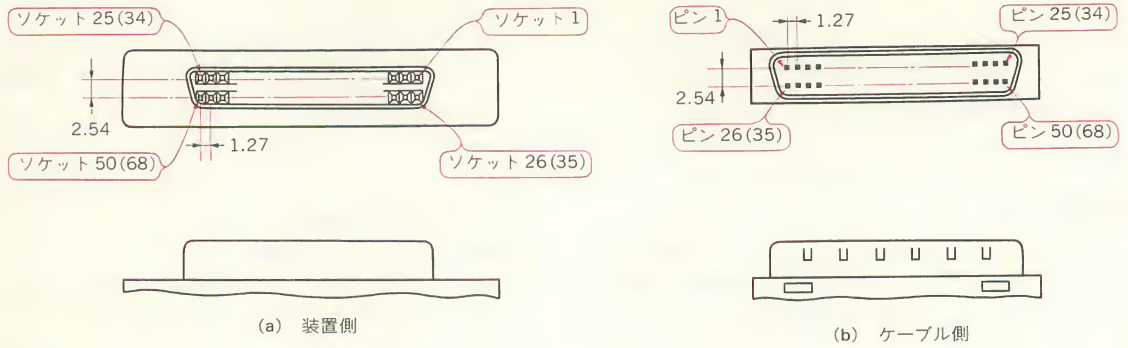
シングル・イニシエータ・オプションの規定は削除されました。したがってイニシエータはセレクション・フェーズ実行時に自分自身とターゲットの SCSI ID をデータ・バス上に出す必要があります。

また、イニシエータはセレクション・フェーズでは必ずアテンション・コンディションを生成する必要があります。つまり通常のシーケンスでは IDENTIFY メッセージを送信してロジカル・ユニットの番号 (LUN) などを通知します。これらは SCSI1 ではオプションとして規定されていたので、これらの機能がサポートされていない SCSI デバイスとの接続を可能にするには、ターゲットとして動作する装置では SCSI1 の規定(イニシエータ ID の通知またはアテンション・コンディションがない場合)でも動作できるように考慮しておく必要があります。

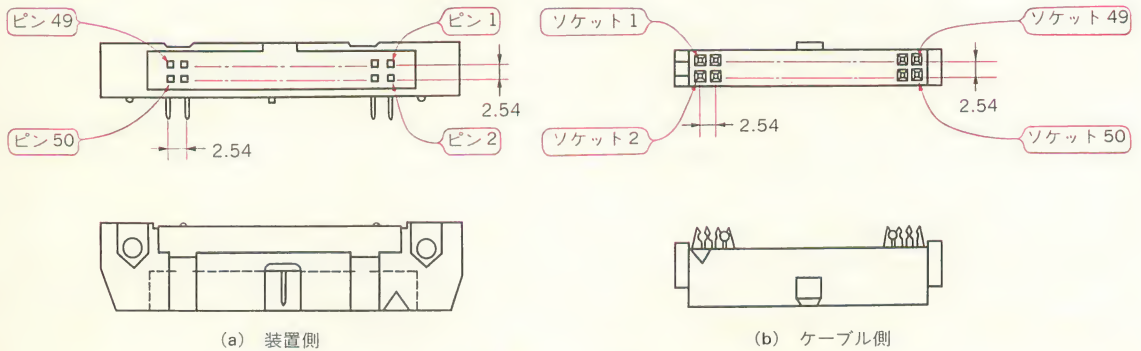
### ● アテンション・コンディション

SCSI1 ではターゲットがアテンション・コンディションに対して応答すべきタイミングが非常に曖昧に記述されていましたが、SCSI2 では各フェーズごとにアテンション・コンディションに対する応答規定が明記されました。ターゲットは少なくともアテンション・コンディションが生成されたバス・フェーズの直後にはメッセージ・アウト・フェーズに移行してイニシエータからのメッセージを受け取らなければなりません。

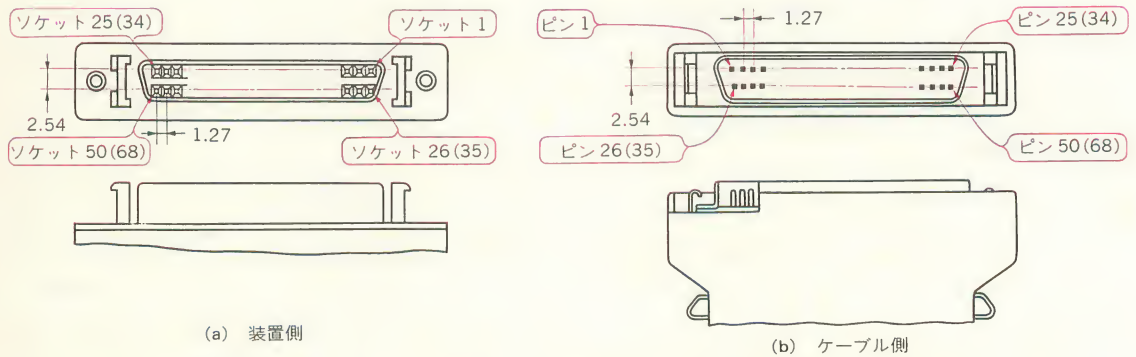
〈図2〉 高密度型ノンシールド・コネクタ (A/B ケーブル)



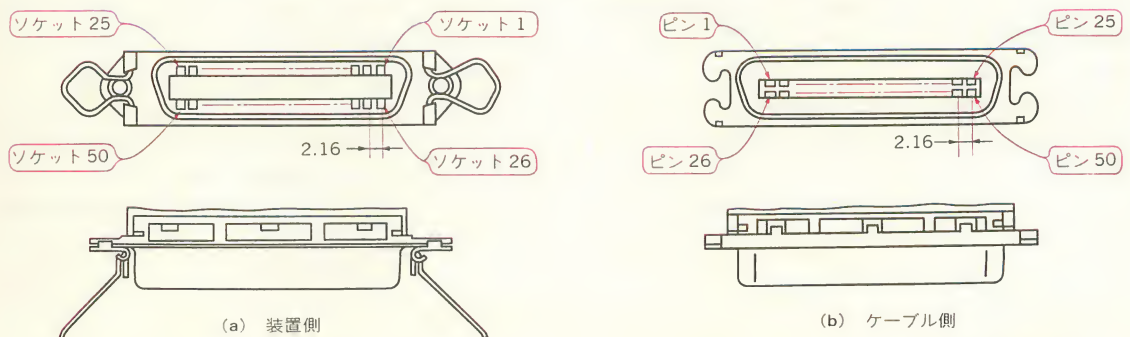
〈図3〉 低密度型ノンシールド・コネクタ (A ケーブル)



〈図4〉 高密度型シールド・コネクタ (A/B ケーブル)



〈図5〉 リボン型シールド・コネクタ (A ケーブル)





〈表3〉 シングル・エンド型バス-A ケーブル信号割り付け

コンタクト番号		信号	信号	コンタクト番号	
01	01	GND	- DB <sub>0</sub>	02	26
02	03	GND	- DB <sub>1</sub>	04	27
03	05	GND	- DB <sub>2</sub>	06	28
04	07	GND	- DB <sub>3</sub>	08	29
05	09	GND	- DB <sub>4</sub>	10	30
06	11	GND	- DB <sub>5</sub>	12	31
07	13	GND	- DB <sub>6</sub>	14	32
08	15	GND	- DB <sub>7</sub>	16	33
09	17	GND	- DB <sub>P</sub>	18	34
10	19	GND	GND	20	35
11	21	GND	GND	22	36
12	23	(リザーブ)	(リザーブ)	24	37
13	25	(オープン)	TERMPWR	26	38
14	27	(リザーブ)	(リザーブ)	28	39
15	29	GND	GND	30	40
16	31	GND	- ATN	32	41
17	33	GND	GND	34	42
18	35	GND	- BSY	36	43
19	37	GND	- ACK	38	44
20	39	GND	- RST	40	45
21	41	GND	- MSG	42	46
22	43	GND	- SEL	44	47
23	45	GND	- C/D	46	48
24	47	GND	- REQ	48	49
25	49	GND	- I/O	50	50

2.54mmピッチ低密度型コネクタ (図3) のコンタクト番号  
1.27mmピッチ高密度型 (図2.4), リボン型シールド・コネクタ (図5) のコンタクト番号

〈表4〉 シングル・エンド型バス-B ケーブル信号割り付け

コンタクト番号	信号	信号	コンタクト番号
01	GND	GND	35
02	GND	-DB <sub>8</sub>	36
03	GND	-DB <sub>9</sub>	37
04	GND	-DB <sub>10</sub>	38
05	GND	-DB <sub>11</sub>	39
06	GND	-DB <sub>12</sub>	40
07	GND	-DB <sub>13</sub>	41
08	GND	-DB <sub>14</sub>	42
09	GND	-DB <sub>15</sub>	43
10	GND	-DB <sub>P1</sub>	44
11	GND	-ACKB	45
12	GND	GND	46
13	GND	-REQB	47
14	GND	-DB <sub>16</sub>	48
15	GND	-DB <sub>17</sub>	49
16	GND	-DB <sub>18</sub>	50
17	TERMPWRB	TERMPWRB	51
18	TERMPWRB	TERMPWRB	52
19	GND	-DB <sub>19</sub>	53
20	GND	-DB <sub>20</sub>	54
21	GND	-DB <sub>21</sub>	55
22	GND	-DB <sub>22</sub>	56
23	GND	-DB <sub>23</sub>	57
24	GND	-DB <sub>P2</sub>	58
25	GND	-DB <sub>24</sub>	59
26	GND	-DB <sub>25</sub>	60
27	GND	-DB <sub>26</sub>	61
28	GND	-DB <sub>27</sub>	62
29	GND	-DB <sub>28</sub>	63
30	GND	-DB <sub>29</sub>	64
31	GND	-DB <sub>30</sub>	65
32	GND	-DB <sub>31</sub>	66
33	GND	-DB <sub>P3</sub>	67
34	GND	GND	68

〈表5〉 ディファレンシャル型バス-A ケーブル信号割り付け

コンタクト番号		信号	信号	コンタクト番号	
01	01	GND	GND	02	26
02	03	+ DB <sub>0</sub>	- DB <sub>0</sub>	04	27
03	05	+ DB <sub>1</sub>	- DB <sub>1</sub>	06	28
04	07	+ DB <sub>2</sub>	- DB <sub>2</sub>	08	29
05	09	+ DB <sub>3</sub>	- DB <sub>3</sub>	10	30
06	11	+ DB <sub>4</sub>	- DB <sub>4</sub>	12	31
07	13	+ DB <sub>5</sub>	- DB <sub>5</sub>	14	32
08	15	+ DB <sub>6</sub>	- DB <sub>6</sub>	16	33
09	17	+ DB <sub>7</sub>	- DB <sub>7</sub>	18	34
10	19	+ DB <sub>P</sub>	- DB <sub>P</sub>	20	35
11	21	DIFFSENS	GND	22	36
12	23	(リザーブ)	(リザーブ)	24	37
13	25	TERMPWR	TERMPWR	26	38
14	27	(リザーブ)	(リザーブ)	28	39
15	29	+ ATN	- ATN	30	40
16	31	GND	GND	32	41
17	33	+ BSY	- BSY	34	42
18	35	+ ACK	- ACK	36	43
19	37	+ RST	- RST	38	44
20	39	+ MSG	- MSG	40	45
21	41	+ SEL	- SEL	42	46
22	43	+ C/D	- C/D	44	47
23	45	+ REQ	- REQ	46	48
24	47	+ I/O	- I/O	48	49
25	49	GND	GND	50	50

2.54mmピッチ低密度型コネクタ (図3) のコンタクト番号  
1.27mmピッチ高密度型 (図2.4), リボン型シールド・コネクタ (図5) のコンタクト番号

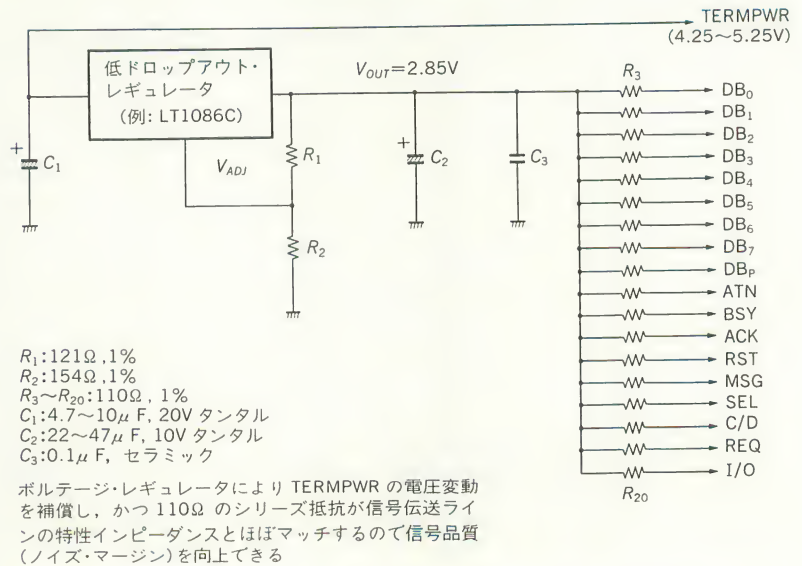
〈表6〉 ディファレンシャル型バス-B ケーブル信号割り付け

コンタクト番号	信号	信号	コンタクト番号
01	GND	GND	35
02	+DB <sub>8</sub>	-DB <sub>8</sub>	36
03	+DB <sub>9</sub>	-DB <sub>9</sub>	37
04	+DB <sub>10</sub>	-DB <sub>10</sub>	38
05	+DB <sub>11</sub>	-DB <sub>11</sub>	39
06	+DB <sub>12</sub>	-DB <sub>12</sub>	40
07	+DB <sub>13</sub>	-DB <sub>13</sub>	41
08	+DB <sub>14</sub>	-DB <sub>14</sub>	42
09	+DB <sub>15</sub>	-DB <sub>15</sub>	43
10	+DB <sub>P1</sub>	-DB <sub>P1</sub>	44
11	+ACKB	-ACKB	45
12	GND	DIFFSENS	46
13	+REQB	-REQB	47
14	+DB <sub>16</sub>	-DB <sub>16</sub>	48
15	+DB <sub>17</sub>	-DB <sub>17</sub>	49
16	+DB <sub>18</sub>	-DB <sub>18</sub>	50
17	TERMPWRB	TERMPWRB	51
18	TERMPWRB	TERMPWRB	52
19	+DB <sub>19</sub>	-DB <sub>19</sub>	53
20	+DB <sub>20</sub>	-DB <sub>20</sub>	54
21	+DB <sub>21</sub>	-DB <sub>21</sub>	55
22	+DB <sub>22</sub>	-DB <sub>22</sub>	56
23	+DB <sub>23</sub>	-DB <sub>23</sub>	57
24	+DB <sub>P2</sub>	-DB <sub>P2</sub>	58
25	+DB <sub>24</sub>	-DB <sub>24</sub>	59
26	+DB <sub>25</sub>	-DB <sub>25</sub>	60
27	+DB <sub>26</sub>	-DB <sub>26</sub>	61
28	+DB <sub>27</sub>	-DB <sub>27</sub>	62
29	+DB <sub>28</sub>	-DB <sub>28</sub>	63
30	+DB <sub>29</sub>	-DB <sub>29</sub>	64
31	+DB <sub>30</sub>	-DB <sub>30</sub>	65
32	+DB <sub>31</sub>	-DB <sub>31</sub>	66
33	+DB <sub>P3</sub>	-DB <sub>P3</sub>	67
34	GND	GND	68

〈表 7〉 SCSI バスの電氣的条件

			SCSI2		SCSI1	
シングル・エンド型バス (最大ケーブル長 6 m)	ドライバ特性	“L” レベル出力電圧: $V_{OL}$	0.0~0.5VDC		0.0~0.4VDC	
		“H” レベル出力電圧: $V_{OH}$	2.5~5.25VDC		2.5~5.25VDC	
		“L” レベル出力電流: $I_{OL}$	48mA (@0.5VDC)		48mA (@0.5VDC)	
	レシーバ特性	“L” レベル入力電圧: $V_{IL}$	0.0~0.8VDC		0.0~0.8VDC	
		“H” レベル入力電圧: $V_{IH}$	2.0~5.25VDC		2.0~5.25VDC	
		“L” レベル入力電流: $I_{IL}$	-0.4~0.0mA (@0.5VDC)		-0.4mA (@0.4VDC)	
		“H” レベル入力電流: $I_{IH}$	0.0~0.1mA (@2.7VDC)		—	
		ヒステリシス: $\Delta V_{TH}$	0.2VDC min.		0.2VDC min.	
		入力容量	25pF max.		—	
	ターミネータ用電源	供給電圧: $V_{TERM}$	4.25~5.25VDC		4.0~5.25VDC	
		電流量	A ケーブル	900mA min.	800mA min.	
			B ケーブル	1500mA min.	—	
ディファレンシャル型バス (最大長 25 m)	ドライバ特性	“L” レベル出力電圧: $V_{OL}$	1.7VDC max. ( $I_{OL}=55mA$ )		2.0VDC max. ( $I_{OL}=55mA$ )	
		“H” レベル出力電圧: $V_{OH}$	2.7VDC min. ( $I_{OH}=-55mA$ )		3.0VDC min. ( $I_{OH}=-55mA$ )	
		ディファレンシャル電圧: $V_{OD}$ (コモン・モード電圧: $-7\sim+12VDC$ )	1.0VDC min.		1.0VDC min.	
	レシーバ特性	入力電流: $I_I$ (入力電圧: $-7\sim+12VDC$ )	$\pm 2.0mA$ max.		$\pm 2.0mA$ max.	
		ヒステリシス: $\Delta V_{TH}$	35mV min.		35mV min.	
		入力容量	25pF max.		—	
		供給電圧: $V_{TERM}$	4.0~5.25VDC		4.0~5.25VDC	
	ターミネータ用電源	電流量	A ケーブル	600mA min.	600mA min.	
			B ケーブル	1000mA min.	—	

〈図 6〉 シングル・エンド型バス用の  
アクティブ・ターミネータ



また、イニシエータに対してはメッセージの種類ごとにアテンション・コンディションを継続してもよいか否かの条件が規定されました(表 9, 表 10)。

### ● FAST SCSI

FAST SCSI は同期モードのデータ転送フェーズでのみ使用できます。SDTR (SYNCHRONOUS DATA TRANSFER REQUEST) メッセージで同期モード転送のパラメータを SCSI デバイス間で定義するとき、転送周期 (Transfer Period) が 200ns より小

さい値、すなわち 5M バイト/s を超える転送速度に決定された場合には FAST SCSI のタイミング規約を適用してデータ転送を実行します。

なお、Transfer Period の最小値は 100ns (=10M バイト/s) です。SDTR メッセージで 200ns 以上の Transfer Period 値が決定されたときには SCSI1 のタイミング規定で動作しなければなりません。

図 7 にデータ転送のタイミング規定を示します。FAST SCSI の転送タイミングは相当に厳しい値なの



〈表 8〉 タイミング規約

名称	規定値	タイミング規定
Arbitration Delay	最小 2.4 $\mu$ s	アービトレーション・フェーズで SCSI デバイスが BSY 信号および SCSI ID を送出してからバス使用权の優先順位を決定するためにデータ・バス上の値を判定するまでの最小待機時間、最大待機時間は規定されない
Assertion Period	最小 90ns	同期モードのデータ転送時における REQ 信号および ACK 信号(B ケーブルについては REQB 信号および ACKB 信号)の最小パルス幅
Bus Clear Delay	最大 800ns	以下のいずれかの事象が発生してから SCSI デバイスがすべてのバス信号の駆動を停止するまでの最大許容時間。 ① バス・フリー・フェーズを検出したとき ② アービトレーション・フェーズを実行中に他の SCSI デバイスが SEL 信号を TRUE にしたことを検出したとき ③ RST 信号が TRUE になったこと(リセット・コンディション)を検出したとき
Bus Free Delay	最小 800ns	SCSI デバイスがバス・フリー・フェーズを検出してからアービトレーション・フェーズを開始するために BSY 信号および SCSI ID を送出するまでの最小待機時間
Bus Set Delay	最大 1.8 $\mu$ s	SCSI デバイスがバス・フリー・フェーズを検出してからアービトレーション・フェーズを開始するために BSY 信号と SCSI ID とを送出するまでの最大許容時間
Bus Settle Delay	最小 400ns	特定の制御信号の状態が変化してからバスの状態が安定するまでの最小待機時間
Cable Skew Delay	最大 10ns	任意の 2 台の SCSI デバイス間で任意のバス信号間のインターフェース・ケーブル上の信号伝播時間の最大許容差
Data Release Delay	最大 400ns	I/O 信号の状態が FALSE から TRUE に変化した後、イニシエータがデータ・バス信号の駆動を停止するまでの最大許容時間
Deskew Delay	最小 45ns	バス信号間の伝送スキュー補償の最小時間
Disconnection Delay	最小 200 $\mu$ s	ターゲットがイニシエータからの DISCONNECT メッセージによりバス・フリー・フェーズに移行した後、つぎにアービトレーションに参加するまでの最小待機時間
Hold Time	最小 45ns	同期モードのデータ転送においてデータを受信する SCSI デバイスでのホールド時間を補償するために REQ 信号または ACK 信号(B ケーブルについては、REQB 信号と ACKB 信号)パルスの前縁からデータ・バス上の転送データを保持していなければならない最小時間
Negation Period	最小 90ns	同期モードのデータ転送において REQ 信号の後縁から次の REQ 信号の前縁まで、または ACK 信号の後縁からつぎの ACK 信号の前縁まで(B ケーブルでは REQB 信号と ACKB 信号)の最小時間
Power-On to Selection Time	最大 10s [推奨値]	電源投入後にターゲットが TEST UNIT READY, INQUIRY, または REQUEST SENSE コマンドに応答できるようになるまでの最大許容時間
Reset Hold Time	最小 25 $\mu$ s	リセット・コンディションを生成するとき RST 信号を TRUE にする最小保持時間、最大時間は規定されない
Reset to Selection Time	最大 10s [推奨値]	ハード・リセット・コンディションが解除された後にターゲットが TEST UNIT READY, INQUIRY, または REQUEST SENSE コマンドに応答できるようになるまでの最大許容時間
Selection Abort Time	最大 200 $\mu$ s	セレクション・フェーズまたはリセレクション・フェーズで SCSI デバイスが自己が選択されていることを認識してから、BSY 信号を応答するまでの最大許容時間
Selection Timeout Delay	最小 250ms [推奨値]	セレクション・フェーズまたはリセレクション・フェーズでイニシエータまたはターゲットがタイムアウト処理を開始するまでに選択対象の SCSI デバイスからの BSY 信号応答を待つ最小時間
Transfer Period	100ns~ 1020ns	同期モードのデータ転送において REQ 信号の前縁から次の REQ 信号の前縁まで、または ACK 信号の前縁からつぎの ACK 信号の前縁まで(B ケーブルでは REQB 信号と ACKB 信号)の最小時間[最小繰返し時間]、具体的な値は SDTR メッセージの交換により、イニシエータとターゲット間で決定される
Fast Assertion Period	最小 30ns	FAST 同期モードのデータ転送時における REQ 信号および ACK 信号(B ケーブルでは REQB 信号および ACKB 信号)の最小パルス幅
Fast Cable Skew Delay	最大 5ns	FAST 同期モード転送を行うときの任意の 2 台の SCSI デバイス間での任意のバス信号間のインターフェース・ケーブル上の信号伝播時間の最大許容差
Fast Deskew Delay	最小 20ns	FAST 同期モード転送を行うときのバス信号間の伝送スキュー補償の最小時間
Fast Hold Time	最小 10ns	FAST 同期モードのデータ転送においてデータを受信する SCSI デバイスでのホールド時間を補償するため、REQ 信号または ACK 信号(B ケーブルについては REQB 信号と ACKB 信号)パルスの前縁からデータ・バス上の転送データを保持していなければならない最小時間
Fast Negation Period	最小 30ns	FAST 同期モードのデータ転送において REQ 信号の後縁から次の REQ 信号の前縁まで、または ACK 信号の後縁からつぎの ACK 信号の前縁まで(B ケーブルでは REQB 信号と ACKB 信号)の最小時間

で、適用に際してはドライバ/レシーバやケーブルの特性に十分注意する必要があります。特にシングル・エンド型バスではケーブル長やシステム構成およびデータ転送速度の上限を制限するなどの考慮が必要になると思われます。SCSI2 の規格書にはシングル・エンド型バスではタイミング・パラメータのテストがされ

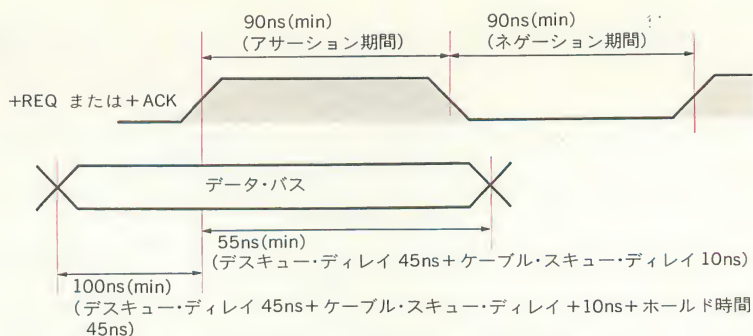
ていないこと、および FAST SCSI は推奨できないことが注記されています。

#### ● WIDE SCSI

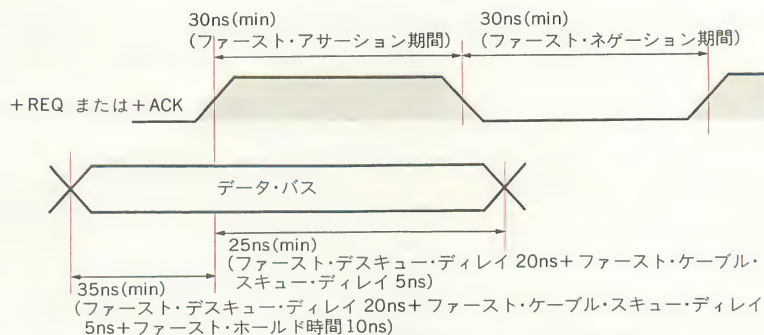
WIDE SCSI は拡張データ・バス(B ケーブル)を使用して A ケーブル上のデータ・バスと並列転送をすることにより、16 ビット幅または 32 ビット幅でのデー

〈図 7〉

同期モードのデータ転送タイミング規定  
(送端側デバイス)



(a) 5M バイト/s以下の時のタイミング：SCS11 と同一



(b) 5M バイト/sを超える時のタイミング：FAST SCSI

タ転送をすることができます。使用するバス幅は 16 ビットまたは 32 ビットのどちらかを選択することができますが、32 ビット転送をサポートするときは 16 ビットでの転送も動作できるようにしておくことが推奨されています。WIDE 転送はデータ・フェーズでのみ使用可能であり、コマンドなどの他の情報転送は A ケーブルの 8 ビットのデータ・バスで実行します。

WIDE 転送を使用するときは、その SCSI デバイス間で WDTR(WIDE DATA TRANSFER REQUEST)メッセージを交換してデータ転送時に使用するデータ・バスの幅を定義しておきます。メッセージ交換の手順は同期モード転送を行うときの SDTR メッセージの交換手順と同様です。WIDE 転送では一度に 2 バイトまたは 4 バイトの転送を行いますが、このときのデータ・バイトの転送順番を図 8 に示します。

WIDE 転送は非同期モードでも同期モードでも実行することができますが、同期モード転送のパラメータ設定は WDTR メッセージによりクリアされることになっているため、同期モードの WIDE 転送を行う場合は WDTR メッセージの交換の後に SDTR メッセージの交換を実行する必要があります。

A ケーブルと B ケーブルでは同一のモード(非同期モードまたは同一転送パラメータでの同期モード)でデータ転送を行いますが、転送タイミングはそれぞれ

に個別の制御線(A ケーブル：REQ/ACK, B ケーブル：REQB/ACKB)で制御されます。したがって、両者のケーブル長などの相違についての制約がなくなり、図 9 に示すように B ケーブルを持った装置と A ケーブルだけの装置を混在して接続することもできます。

2 本のケーブル間での転送タイミングの規定はありませんが、ターゲットは REQ と REQB の発行タイミングを、イニシエータは ACK と ACKB の応答タイミングを制御することで、A/B ケーブルの転送を同期させることができます。データ転送の正常性を保証するため、ターゲットはひとつのデータ・フェーズでの REQ/ACK ハンドシェイク数と REQB/ACKB ハンドシェイク数が等しいことを確認します。

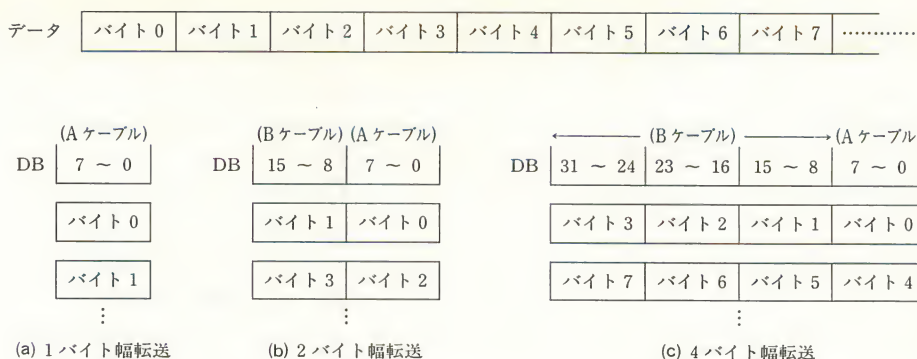
転送データのバイト数が 2 または 4 の倍数でないために転送の最後のデータがバス幅に満たないときは残りのバイト位置にはどんな値を送出しても構いませんが、パリティの値は正しくなければなりません。データ・イン・フェーズでバス幅に満たない転送をしたときはターゲットは **IGNORE WIDE RESIDUE** メッセージをデータ・フェーズの直後に送信して無効なデータ・バイト数をイニシエータに通知します。

### ● バス・フェーズのシーケンス

アービトラージョン・フェーズとリセクション・フェーズ、およびセクション・フェーズでのアテンション・コンディションが標準機能になったので、バ



〈図8〉  
WIDE SCSI での  
データ転送順序



先頭データ (MSB) を A ケーブル : DB<sub>7</sub>~DB<sub>0</sub>から転送する

ス・フェーズの推移シーケンスは図 10 に示すようになります。

## メッセージ・システム

メッセージ・システムはインターフェースの物理レイヤの管理を目的としていましたが、SCSI2ではコマンド・キューイングなどの論理階層も制御するように仕様が拡張され、新たに2バイト長のメッセージなどが追加されました。表9と表10にSCSI2で定義されているメッセージの一覧を示します。メッセージ・プロトコルの曖昧さを避けるため、イニシエータが送信するいくつかのメッセージについては、そのメッセージの送信終了時にアテンション・コンディションを解除しなければならないことが規定されました。

### ● 機能が拡張されたメッセージ

#### ▶ DISCONNECT (04h)

イニシエータからもこのメッセージを発行できるようになりました。従来ディスクコネクト処理はターゲットが決定するタイミングでのみ可能でしたが、この機能拡張によりイニシエータはアテンション・コンディションを生成してターゲットにディスクコネクトを要求することができます。

なお、このメッセージの指示でディスクコネクトを実行したターゲットは200μs(Disconnection Delay)を経過するまでつぎのアービトレーションを開始できません。

#### ▶ IDENTIFY (80h~F7h)

ターゲット・ルーチンと呼ばれる機能が規定され、このメッセージは図11に示すような定義になりました。ターゲット・ルーチンについては保守や診断を目的としてターゲット上(ロジカル・ユニットではない)で実行される機能で、INQUIRYあるいはREQUEST SENSEコマンドとのみ組み合わせて使用します。最大8つまでのターゲット・ルーチンを定義できるとされていますが、具体例は明らかではありません。

ません。IDENTIFYメッセージでLUNTRNビットに“1”を指定することによってターゲット・ルーチンを起動します。この場合、LUNTRNフィールドはロジカル・ユニット番号ではなく、ターゲット・ルーチンの種別を指定することになります。

### ● コマンド・キューイング用のメッセージ

タグ付(Tagged)コマンド・キューイングの仕組みについては後で詳細に説明しますが、以下の5種類のメッセージをサポートする必要があります。

最初の3つのメッセージ(キュー・タグ・メッセージと呼ぶ)は図12に示す構造を持つ2バイト長のメッセージで、イニシエータが発行する個々のコマンドに固有な識別子(キュー・タグ)を与えるためのものです。キュー・タグは符号なしの2進数で、最大256コマンドまでの識別が可能です。

#### ▶ HEAD OF QUEUE TAG (21h)

このメッセージで発行されたコマンドはLIFO (Last-In, First-Out)の実行順序となるようにコマンド・キューの先頭に置かれ、実行中のコマンドが終了すると最優先で実行されます。

#### ▶ ORDERED QUEUE TAG (22h)

このメッセージはコマンドを到着順に実行することを指定します。このメッセージで発行されたコマンドは、すでにキューイングされている他のすべてのコマンドが終了した後に行われ、また、そのコマンドの後に到着したコマンドは、HEAD OF QUEUE TAGメッセージで発行されたコマンドを除いてそのコマンドの後に実行されます。

#### ▶ SIMPLE QUEUE TAG (20h)

このメッセージで発行されたコマンドについてはターゲットが実行順序を変更することができます。一般にターゲットはキューイングされているコマンド列を性能が最適化されるような順序で実行しますが、実行順序の変更範囲を制約できるようにするためのモード・セレクト・パラメータも用意されています。

また、ターゲットはタグ付コマンドのリコネクション

ンを実行するときにこのメッセージでコマンドのキュー・タグをイニシエータに通知します。

#### ▶ ABORT TAG (0Dh)

このメッセージは指定したロジカル・ユニットで実行中またはキューイング中の**特定のひとつのコマンド** (このメッセージを発行したイニシエータが発行したコマンド)のみをクリアします。

SCSIパス上で実行中のコマンドはこのメッセージだけでクリアできますが、ディスク接続中のコマンドをクリアする場合にはこのメッセージに先立ってロジカル・ユニット番号 (IDENTIFY) とクリアするコマンドのキュー・タグ (SIMPLE QUEUE TAG) を指定します。

#### ▶ CLEAR QUEUE (0Eh)

このメッセージは指定したロジカル・ユニットで実行中またはキューイング中の**すべてのコマンド** (どのイニシエータが発行したコマンドにかかわらず) をクリアします。

このメッセージを発行したイニシエータ以外のイニシエータからのコマンドがクリアされた場合にはそれらのイニシエータに対してユニット・アテンション・コンディションが発生します。

#### ● WIDE SCSI 用のメッセージ

WIDE SCSI のために必要なメッセージとしてつぎのふたつがあります。

#### ▶ WDTR (拡張メッセージ: 03h)

イニシエータとターゲット間でこのメッセージを交換し、**データ転送に使用するバスの幅を決定**するものです (図 13)。

#### ▶ IGNORE WIDE RESIDUE (23h)

データ・イン・フェーズの最後のバイト数がバス幅に満たないとき、ターゲットからイニシエータに対して**無効なバイト数を通知**するために使用します (図 14)。

#### ● ECA (Extended Contingent Allegiance) 用のメッセージ

ECA 状態の規定については後で説明しますが、その制御用としてつぎのふたつのメッセージがあります。

#### ▶ INITIATE RECOVERY (0Fh)

このメッセージは **ECA 状態の開始を宣言**するもので、通常はターゲ

ットからイニシエータに対して発行されます。例外として AEN プロトコルのときに一時的にイニシエータとして動作しているデバイス (通常はターゲット) がこのメッセージを発行して ECA 状態に入ることがあります。

#### ▶ RELEASE RECOVERY (10h)

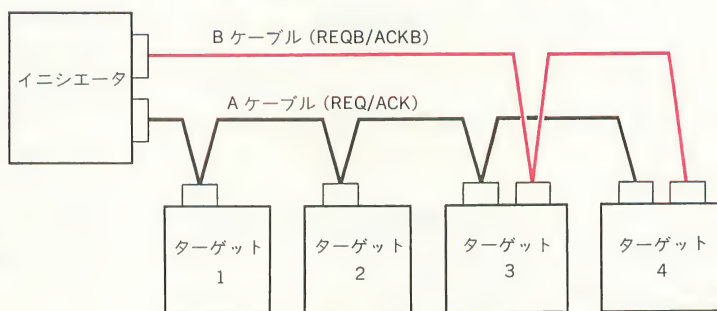
このメッセージはイニシエータからターゲットに対して **ECA 状態の解除**を指示するものです。

#### ● その他の追加メッセージ

イニシエータから実行中のコマンドを途中で終了させる (それまでの実行結果は保証して終了させる) 手段として **TERMINATE I/O PROCESS** メッセージが用意されました。

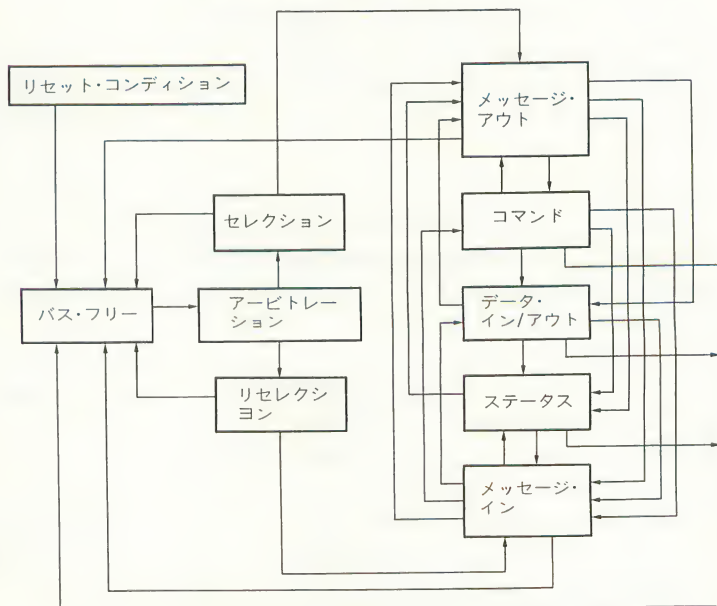
このメッセージを受信したターゲットは可能な限り

〈図 9〉 A ケーブルと A/B ケーブル (WIDE SCSI) の混在接続



A ケーブルと B ケーブルのデータ転送はそれぞれに専用の制御線 (REQ/ACK, REQB/ACKB) でハンドシェイクされるので A ケーブルだけの装置と A/B ケーブルをもった装置を混在して接続できる。ケーブル長は異なってもよい

〈図 10〉 バス・フェーズの推移シーケンス





〈表 9〉メッセージ一覧

メッセージ・コード	メッセージ名称	ATN 解除 条件	方向	サポートの要否			
				SCSI2		CCS	SCSI1
				I	T		
00h	COMMAND COMPLETE	—	IN	M	M	M	M
01h	拡張メッセージ(表 10)	—	—	—	—	—	—
02h	SAVE DATA POINTER	—	IN	O	O	O	O
03h	RESTORE POINTERS	—	IN	O	O	O	O
04h	DISCONNECT	—	IN	O	O	O	O
		YES	OUT	O	O	—	—
05h	INITIATOR DETECTED ERROR	YES	OUT	M	M	O	O
06h	ABORT	YES	OUT	0	M	M	O
07h	MESSAGE REJECT	—	IN	M	M	M	O
		YES	OUT	M	M	M	O
08h	NO OPERATION	YES	OUT	M	M	M	O
09h	MESSAGE PARITY ERROR	YES	OUT	M	M	O	O
0Ah	LINKED COMMAND COMPLETE	—	IN	O	O	O	O
0Bh	LINKED COMMAND COMPLETE WITH FLAG	—	IN	O	O	O	O
0Ch	BUS DEVICE RESET	YES	OUT	O	M	M	O
0Dh	ABORT TAG	YES	OUT	O	O	R	R
0Eh	CLEAR QUEUE	YES	OUT	O	O	R	R
0Fh	INITIATE RECOVERY	—	IN	O	O	R	R
		YES	OUT	O	O	R	R
10h	RELEASE RECOVERY	YES	OUT	O	O	R	R
11h	TERMINATE I/O PROCESS	YES	OUT	O	O	R	R
12h~1Fh	(1 バイト長メッセージ)	—	—	R	R	R	R
20h	SIMPLE QUEUE TAG	—	IN	O	O	R	R
		NO	OUT	O	O	R	R
21h	HEAD OF QUEUE TAG	NO	OUT	O	O	R	R
22h	ORDERED QUEUE TAG	NO	OUT	O	O	R	R
23h	IGNORE WIDE RESIDUE	—	IN	O	O	R	R
24h~2Fh	(2 バイト長メッセージ)	—	—	R	R	R	R
30h~7Fh		—	—	R	R	R	R
80h~FFh	IDENTIFY	—	IN	M	O	O	O
		NO	OUT	M	M	O	M(T)

I：イニシエータ，T：ターゲット，  
M：サポート必須，O：オプション，R：リザーブ  
IN：ターゲット→イニシエータ，OUT：イニシエータ→ターゲット

表中の ATN 解除条件の欄が“YES”のメッセージをイニシエータが送信するとき、イニシエータはそのメッセージ・アウト・フェーズの最後の ACK 信号応答の 90ns(Deskew Delay×2)以上前に ATN 信号を FALSE にし、アテンション・コンディションを解除しなければならない。この条件に違反したとき、ターゲットはプロトコル上のエラーとみなしてバス・フリー・フェーズに移行する

〈表 10〉拡張メッセージ一覧

拡張 メッセージ・ コード	メッセージ 長 (バイト)	メッセージ名称	ATN 解除 条件	方向	サポートの要否			
					SCSI2		CCS	SCSI1
					I	T		
00h	7	MODIFY DATA POINTER	—	IN	O	O	O	O
01h	5	SYNCHRONOUS DATA TRANSFER REQUEST	—	IN	O	O	O	O
			YES	OUT	O	O	O	O
02h	—	(EXTENDED IDENTIFY)⇒削除	—	—	R	R	O	O
03h	4	WIDE DATA TRANSFER REQUEST	—	IN	O	O	O	O
			YES	OUT	O	O	O	O
04h~7Fh	—		—	—	R	R	R	R
80h~FFh	—		—	—	V	V	V	V

I：イニシエータ，T：ターゲット，  
O：オプション，V：ベンダ固有，R：リザーブ  
IN：ターゲット→イニシエータ，OUT：イニシエータ→ターゲット  
ATN 解除条件：表 9

〈図 11〉  
IDENTIFY メッセージ

ビット	7	6	5	4	3	2	1	0
バイト 0	1	DiscPriv	LUNTAR	0	0	LUNTRN		
				LUNTAR	LUNTRN			
				0	ロジカル・ユニット番号(LUN)			
				1	ターゲット・ルーチン番号			

〈図 12〉  
キュー・タグ・メッセージ

ビット	7	6	5	4	3	2	1	0
バイト 0	メッセージ・コード：20h, 21h, または 22h							
バイト 1	キュー・タグ：00h~FFh							

キュー・タグ・メッセージはアイデンティファイ・メッセージに続けて発行され、コマンドの識別子(キュー・タグ)を宣言する

早い時点でコマンドの実行を打ち切り、特別なステータス(COMMAND TERMINATED)でそのコマンドを終了させます。このときターゲットはセンス・データを生成し、そのコマンドがデータ転送をとまなうものであれば実行停止時まで正常に転送を完了したデータ量をインシエータが算出できるような情報をセンス・データに表示します。

その他に ANSI ではディスク・アレイなどのアプリケーションを意図して、ターゲットとのデータ転送の開始タイミングをインシエータから制御するためのメッセージを検討していますが、SCSI2 の最終規格に組み込まれるか否かは明らかではありません。

#### ● 削除されたメッセージ

ロジカル・ユニット番号(LUN)を拡張するために定義されていた EXTENDED IDENTIFY メッセージ(拡張メッセージ・コード：02h)が削除されました。

### 共通論理仕様

すべてのデバイス・タイプに共通な論理仕様で、規定が変更されたり機能が追加された主要な項目を以下に説明します。

#### ● コマンド(CDB: Command Descriptor Blok)の記述規定

##### ▶ グループ 2 の CDB

10 バイト長の CDB としてグループ 2(オペレーション・コード：40h~5Fh)が使用されるようになりました。

##### ▶ LUN フィールド

ロジカル・ユニット番号(LUN)は、必須機能となった INENTIFY メッセージで指定されるので CDB 上の LUN フィールドは意味を持たなくなりましたが、SCSI1 との整合性を保つため従来どおりに定義が残されました。ターゲットはこのフィールドに指定された値は無視します。なお、このフィールドは将来の規格拡張で別の機能に使用される可能性があるため、ゼロ

を指定しておくことが推奨されています。

#### ● ステータス・バイト

図 15 にステータス・バイトの定義を示します。ビット 6, 5, 0 は SCSI1 ではベンダ固有に使用できましたが、それが許されなくなりました。

また、機能が追加にともなって以下のステータス・コードが追加されました。

##### ▶ COMMAND TERMINATED

このステータスは TERMINATE I/O メッセージを受け取ったことによってコマンドの実行が終了したことを示します。

##### ▶ QUEUE FULL

このステータスはコマンド・キューに空きがなく、新たなタグ付コマンドを受け取ってキューイングできないことを示します。

#### ● プログラマブル動作モード

CHANGE DEFINITION コマンド(オプション)を用いて、ターゲットあるいはロジカル・ユニットの動作モード(SCSI の論理仕様レベル)をインシエータが指定できるようになりました。インシエータは SCSI1, CCS, または SCSI2 のいずれかを指定することができます。

ターゲットはコマンド・セットやパラメータの定義、コマンドの動作などを指定された SCSI レベルに変更しますが、その他のパラメータ(ベンダ ID, デバイス・モデル名など)をベンダ固有の規定で変更することも許されています。

インシエータは INQUIRY コマンドで、ターゲットとロジカル・ユニットの現在の動作モードやサポートされている機能レベルを知ることができます。

この機能は特定の SCSI 仕様に依存しているアプリケーションを SCSI2 の環境下で共存させることが必要ときに有用な手段になります。

#### ● AEN(非同期事象の通知)機構

ロジカル・ユニットで非同期に発生した事象をターゲットからインシエータに通知するための機能として



〈図 13〉  
WDTR(WIDE DATA TRANSFER  
REQUEST)メッセージ

ビット	7	6	5	4	3	2	1	0
バイト 0	メッセージ・コード：01h(拡張メッセージ)							
バイト 1	拡張メッセージ長：02h							
バイト 2	拡張メッセージ・コード：03h							
バイト 3	転送バイト幅： $m(2^m)$							

バイト 3 の値がデータ転送時のデータ・バス幅を示す

$m=00h$ ：8 ビット幅

01h：16 ビット幅

02h：32 ビット幅

〈図 14〉  
IGNORE WIDE RESIDUE  
メッセージ

ビット	7	6	5	4	3	2	1	0
バイト 0	メッセージ・コード：23h							
バイト 1	無効データ・バイト数							

バイト 1 の値はデータ・イン・フェーズの最後の REQ(B)/ACK(B)ハンド  
シェイクでの無効な転送バイト数を示す

● 無効なデータ・バイト

バイト 1 の値	32 ビット転送	16 ビット転送
00h	—	—
01h	DB <sub>31</sub> ～DB <sub>24</sub>	DB <sub>15</sub> ～DB <sub>8</sub>
02h	DB <sub>31</sub> ～DB <sub>16</sub>	—
03h	DB <sub>31</sub> ～DB <sub>8</sub>	—

AEN (Asynchronous Event Notification)機構が設け  
られました。

SCSI II ではこの機能がなかったので、ロジカル・ユ  
ニットの状態遷移などを検出するにはイニシエータか  
らのポーリングが必要でした。

代表的な非同期事象にはつぎのものがありますが、  
ベンダ固有の追加定義もできます。

- ・I/O 装置の状態遷移(ノット・レディ→レディ、媒体  
交換やオペレータ操作など)

- ・I/O 動作の終了(MT 装置のリワインドなど)

- ・正常終了報告済みのコマンドでのエラー(ライト・バ  
ック・キャッシュなど)

- ・ユニット・アテンション・コンディション

AEN を使用するターゲットにはイニシエータとし  
て動作できる機能が必要です。

また、AEN の通知先はプロセッサ・デバイスに限  
定されています。

つまり、AEN を使用するイニシエータは、一時的  
にターゲット(プロセッサ・デバイス)として動作でき  
なければなりません。

AEN 機能を備えたターゲットは初期化(電源投入  
時など)時に以下の手順でどの SCSI デバイスが AEN  
受信機能を持っているかを調べ、その SCSI デバイス  
に対してのみ AEN を実行します。

- ① 自分以外のすべての SCSI ID についてセレクシ  
ョン・フェーズを実行して、応答した SCSI デバイ  
スに対し、LUN=0 を指定して INQUIRY コマン  
ドを発行する。
- ② INQUIRY データ上のデバイス・タイプがプロセ

ッサ・デバイスで、かつ AEN の受信が可能である  
こと(AENC ビット=1)が表示されていればテスト  
・ユニット・レディ・コマンドを発行する。

- ③ 必要に応じて REQUEST SENSE, TEST  
UNIT READY を繰り返し、TEST UNIT  
READY コマンドが GOOD ステータスで終了すれ  
ばその SCSI デバイスは AEN を受信できるものと  
判断する。

AEN の事象が発生したとき、そのターゲットは一  
時的にイニシエータとして動作し、送信先の SCSI デ  
バイスの LUN=0 を選択して SEND コマンドを発行  
します。この SEND コマンドでセンス・データと同一  
形式のデータを転送することにより、発生した非同期  
事象の詳細を通知します(図 16)。

AEN の事象が ECA 状態の生成を必要とするとき  
は SEND コマンドのコマンド・フェーズの前に INI  
TIAL RECOVERY メッセージを送信することで  
ECA 状態に入ることができます。

● CA (Contingent Allegiance) 状態

CA 状態はエラー・リカバリのための情報であるセ  
ンス・データの保持条件を明確にしたものです。

CA 状態はターゲットが CHECK CONDITION ス  
テータスまたは COMMAND TERMINATED ステ  
ータスを報告したとき、あるいはターゲットが強制的  
にバス・フリー・フェーズに移行(SCSI バス・プロトコ  
ル上の重度なエラーを検出したとき)してセンス・デー  
タを生成したとき、そのときのイニシエータ・ターゲ  
ット-ロジカル・ユニット(またはターゲット・ルーチ  
ン)のバスに対して発生し、同一のバスでつぎのコマ

〈図 15〉 ステータス・バイト

ビット	7	6	5	4	3	2	1	0
バイト0	R	R	ステータス・バイト・コード					R

R : リザーブ (=0)

ビット6,5,0はベンダ固有に定義できなくなった

ビット	54321
00000	GOOD
00001	CHECK CONDITION
00010	CONDITION MET
00100	BUSY
01000	INTERMEDIATE
01010	INTERMEDIATE - CONDITION MET
01100	RESERVATION CONFLICT
10001	COMMAND TEAMNATED
10100	QUEUE FULL

ンドが発行されるかハード・リセット・コンディション、ABORT メッセージ、または BUS DEVICE RESET メッセージによって解除されます。

通常は、イニシエータが REQUEST SENSE コマンドを発行してセンス・データを読み出します。

CA 状態のとき、ターゲットはそのパスに対してのセンス・データを保持し続けます。ターゲットが他のパスに対して個別にセンス・データを保持できないのであれば、そのターゲットは CA 状態が解除されるまで他のパスのアクセスに対して BUSY ステータスを応答します(図 17)。

#### ● ECA(Extended Contingent Allegiance)状態

CA 状態がセンス・データの排他的保持だけの機能であるのに対し、ECA 状態はエラー・リカバリ処理のための完全な排他制御の手順を提供します。

この機能によってエラー終了したコマンドの後処理やキューイングされているコマンドの処置などの一連のエラー・リカバリ処理を行う間、イニシエータはそのロジカル・ユニットを排他的に占有することができます(図 18)。

ECA 状態は CHECK CONDITION ステータスや COMMAND TERMINATED ステータスの直後(COMMAND COMPLETE メッセージの前)あるいは AEN プロトコルの SEND コマンドの前にターゲット(AEN のときはイニシエータとして動作している)が INITIAE RECOVERY メッセージを発行することで発生し、そのイニシエータが RELEASE RECOVERY メッセージを発行するか、ハード・リセット・コンディションまたは BUS DEVICE RESET メッセージが発行されるまで継続します。ECA 状態にあるターゲットは、そのロジカル・ユニットに対する他のイニシエータからのアクセスに対しては BUSY ステータスを応答します。

ECA 状態はターゲットの宣言によってのみ生成さ

れます。CA 状態と ECA 状態のどちらを発生させるかは発生したエラー状態などによってターゲットが決めることになりますが、その基準は SCSI2 規格の中には規定されていません。

#### ● コマンド・キューイング

コマンド・キューイングにはタグなし(Untagged)キューイングとタグ付き(Tagged)キューイングの2種類の方法がありますが、SCSI2 で“コマンド・キューイング”というときには一般にタグ付きキューイングを指します。

ターゲットは両方のキューイング機能をサポートできます。ひとつのイニシエータはある一時点ではいずれか一方のキューイング手法のみを使用できますが、マルチ・イニシエータのときは他のイニシエータは別のキューイング手法を使用してもよいのでターゲットは両方のキューイングの混在制御が必要になることもあります。

##### ▶ タグなしキューイング

ターゲットはロジカル・ユニット上で実行中のコマンドがあっても他のイニシエータからそのロジカル・ユニットに発行されたコマンドを受け付けてキューイングします。

キューイング可能なコマンドは各ロジカル・ユニットについてイニシエータあたりひとつまでです。

イニシエータはあるロジカル・ユニットに対して発行したコマンドが終了するまでは同一のロジカル・ユニットに別のコマンドを発行することはできません。このキューイング手法は BUSY ステータスによるコマンド再発行のオーバ・ヘッド時間、およびターゲットがつぎのコマンドを開始するまでのオーバ・ヘッド時間などを削減させることができますが、マルチイニシエータのときにしか効果がありません。

##### ▶ タグ付きキューイング

ターゲットは各ロジカル・ユニットについて同一ま



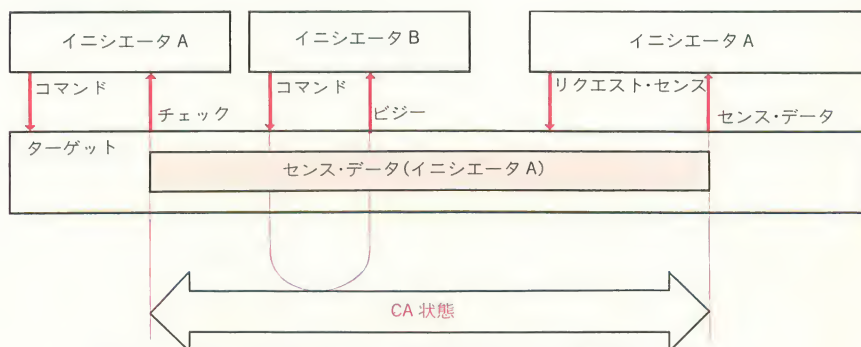
〈図 16〉  
AEN データ形式  
(SEND コマンド)

ビット	7	6	5	4	3	2	1	0					
バイト 0	(リザーブ：00)		LUNTAR	(リザーブ：00)		LUNTRN							
1	(リザーブ：00h)												
2	(リザーブ：00h)												
3	(リザーブ：00h)												
4	センス・データ・バイト(0) . . . . センス・データ・バイト(n)												
~									~	(図 20 と同一形式)		~	
~									~	非同期事象の詳細を表す		~	
n+4													~

(図 20 と同一形式)  
非同期事象の詳細を表す

LUNTAR	LUNTRN	非同期事象の発生元
0	LUN 番号	ロジカル・ユニット
1	ターゲット・ルーチン番号	ターゲット・ルーチン

〈図 17〉 CA 状態



イニシエータ A に対してセンス・データが生成されると CA 状態が発生する。ターゲットが他のイニシエータ用に個別にセンス・データを保持できないのであれば、CA 状態の時にイニシエータ B からコマンドを受け取った時は BUSY ステータスを応答する。ターゲットは CA 状態の間、他のコマンドを実行しないのでイニシエータ A 用のセンス・データは保持され続ける。イニシエータ A がセンス・データを読み出すと CA 状態は解除される。

たは他のイニシエータから発行された複数個のコマンドを受け付けてキューイングします。

仕様上はイニシエータ・ロジカル・ユニットごとに最大 256 個までのコマンドのキューイングが可能です。実際にキューイング可能なコマンドの数はイニシエータおよびターゲットの組み込み仕様に依存します。

イニシエータはコマンド発行時に IDENTIFY メッセージに続けていずれかのキュー・タグ・メッセージを発行し、各コマンドに固有な識別子(キュー・タグ)を与えます。イニシエータはその時点でそのロジカル・ユニット上で実行中またはキューイング中のコマンド(そのイニシエータが発行したコマンド)のキュー・タグと重複しない値を指定します。

イニシエータ内の SCSI ポインタ機構ではキューイングさせるコマンド数分のセーブ・ポインタを用意し、ターゲット ID/LUN/キュー・タグの値に基づいてポインタのセーブやリストアを行います。

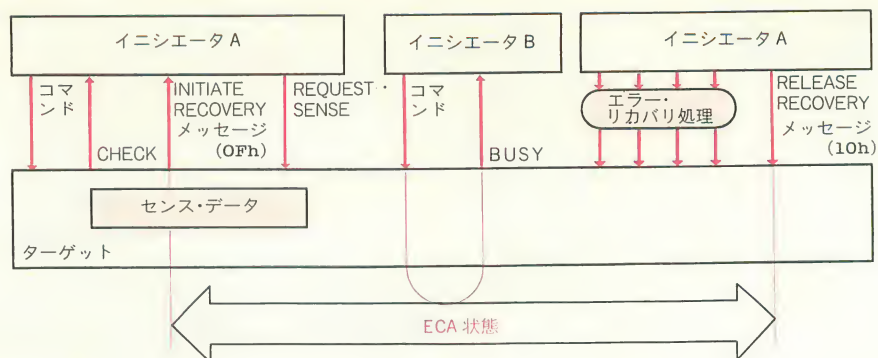
ターゲットはコマンド・キューをロジカル・ユニット単位に管理し、コマンド・キューに空きがなくなったときは新たなタグ付きコマンドに対してキュー・フ

ル・ステータスを応答します。また、タグ付きコマンドのリコネクションを実行するときにはターゲットはリコネクション・シーケンスで IDENTIFY メッセージに続けて SIMPLE QUEUE TAG メッセージを発行し、リコネクションが必要なコマンドのキュー・タグをイニシエータに通知します。

図 19 にタグ付キューイングの実行例を示します。SIMPLE QUEUE TAG メッセージで発行されたコマンドはターゲットがその実行順序を変更できます。ハードディスク装置では一般にリード/ライト・ヘッドの移動が最小になるようにコマンドを並べ変えます(このためシーク・リオーダリングと呼ばれることがある)。ORDERED QUEUE TAG メッセージで発行されたコマンドは必ず到着順に、また、HEAD OF QUEUE TAG メッセージで発行されたコマンドは実行中のコマンドの終了後に最優先で実行されます。

コマンドがエラー終了した場合、そのロジカル・ユニットにキューイングされている後続のコマンドをターゲットがどう扱うかについてはつぎのふたつの方法が規定されており、モード・セレクト・パラメータで指

〈図 18〉 ECA 状態



ターゲットが INITIATE RECOVERY メッセージを送信するとイニシエータ A に対する ECA 状態が発生する。ECA 状態の時に他のイニシエータ(図では、イニシエータ B) からコマンドを受け取った時はターゲットは BUSY ステータスを応答する。イニシエータ A は一連のエラー・リカバリ処理の間、そのターゲットとロジカル・ユニットを占有できる。イニシエータ A が RELEASE RECOVERY メッセージを発行すると ECA 状態は解除される。

定することができます。

- ① キューイングされているコマンドはそのまま保持し、イニシエータにその処置を委ねる
  - ② CA または ECA 状態が解除されたとき、キューイングされているすべてのコマンドをクリア
- どちらの場合も **CA または ECA 状態の間はターゲットはキューイングされている後続のコマンドの実行は開始しません**。また、イニシエータはエラー・リカバリ処理のためのコマンドはタグなしで実行します。

①の方法では ECA 状態が生成されていればイニシエータはコマンド・キュー内のコマンドを選択的にクリアすることなどができます。SCSI2 ではエラー・リカバリ系や例外条件処理の細部までは規定されていませんが、今後タグ付きキューイング機能が一般化する過程で、徐々に標準形が確立されていくと思われます。

なお、タグ付きキュー・イングをサポートしていないターゲットがキュー・タグ・メッセージを受け取ったときは、ターゲットはそのメッセージをリジェクトして処理を継続し、タグなしコマンドとして実行します。

#### ● オーバ・ラップ・コマンド

コマンドの実行が終了する前に同一のイニシエータが同一のロジカル・ユニットに対してつぎのコマンドを発行(タグ付きキューイングのときは同じキュー・タグのコマンド)したとき、またはひとつのイニシエータがタグ付きコマンドとタグなしコマンドを混在して発行しようとしたとき(タグ付きキューイングでの CA または ECA 状態の場合を除く)はオーバラップ・コマンドになります。オーバラップ・コマンドが発生したときターゲットは、そのイニシエータがそのロジカル・ユニットに発行していたすべてのコマンド(実行中あるいはキューイング中)をアボートします。

### 共通コマンドの仕様

すべてのデバイス・タイプに共通なコマンドの一覧を表 11 に示します。SCSI2 では **TEST UNIT READY, REQUEST SENSE, INQUIRY, SEND DIAGNOSTIC** の 4 つのコマンドのサポートが必須になっています。

追加コマンドおよび機能が拡張された主なコマンドの概要を以下に説明します。

#### ● REQUEST SENSE : 03h

CA や ECA 状態が発生したとき、イニシエータは直ちにこのコマンドを発行してエラー・リカバリのための情報(センス・データ)を読み出します。

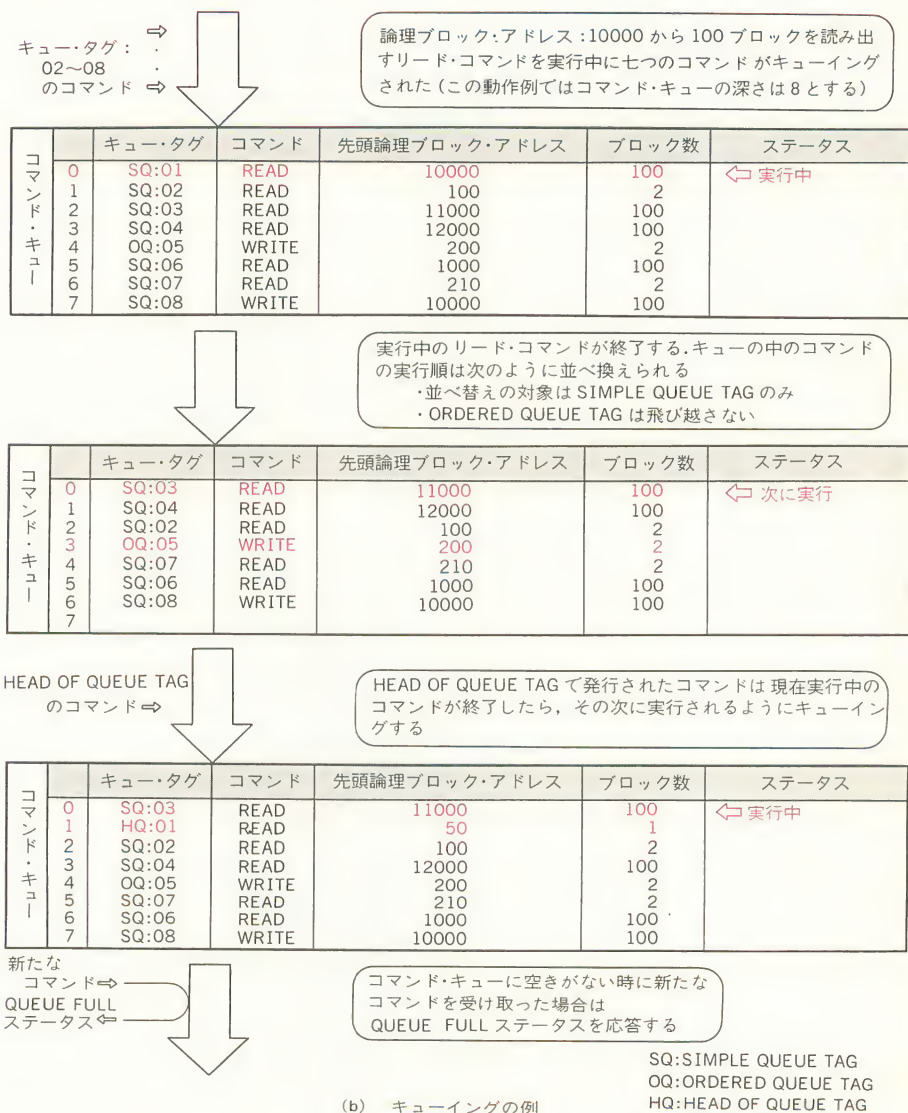
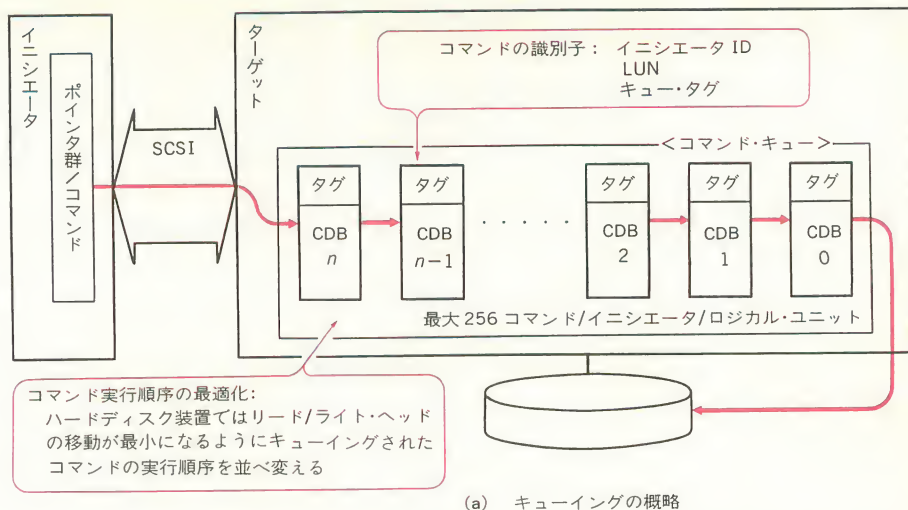
SCSI1 および CCS ではこのコマンドの CDB で転送バイト長(Allocation Length)にゼロが指定されたとき、ターゲットはセンス・データの先頭 4 バイトを転送することになっていましたが、SCSI2 の仕様ではターゲットは何も転送せずにコマンドを終了させ、センス・データをクリアします。

センス・データは図 20 に示す形式(従来、Extended センスと呼ばれていた形式)が標準になり、SCSI1 で許容されていた 4 バイト長のセンス・データ形式は削除されました。ターゲットは少なくとも **18 バイト長のセンス・データをサポートする必要があります**。エラー・コード : 70h のセンス・データは従来と同様に CA または ECA 状態が発生したときのコマンドに関するエラー情報を示します。

一方、エラー・コード : 71h のセンス・データは **デファード・エラー**と呼ばれ、すでにイニシエータに GOOD ステータスを報告したコマンド(イミディエート・ピットの指定で実動作の完了前に正常終了を報告したときやライト・バック・キャッシュなど)の実行中



図 19  
タグ付きコマンド・  
キューイング



にエラーが発生したときに、そのエラー情報を報告するものです。

デファード・エラーは AEN で通知することができますが、AEN がサポートされていないときには同一インシエータからのつぎのコマンドに対して CHECK CONDITION ステータスを通知します。

センス・データのバイト 12～13 の ASC と ASCQ はエラーの発生原因や例外条件を分類してコード化したもので、すべてのデバイス・タイプについて体系化されています。インシエータはほとんどの場合、センス・キー/ASC/ASCQ によりエラー・リカバリ手順を選択することができます。

#### ● INQUIRY : 12h

インシエータはこのコマンドでターゲットおよびロジカル・ユニットの属性や SCSI のレベルおよびサポートされている機能などを知ることができます。

#### ▶ スタンダード INQUIRY データ

このデータはデバイス・タイプなどの装置の一般属性を報告します。図 21 に示すように CCS の規定と同様のデータ形式になっていますが、オプション機能のサポート・レベルを表示するフィールドが追加されています。

ターゲットは少なくともバイト 0～35 までの 36 バイト長の INQUIRY データをサポートする必要があります。

#### ▶ VPD(Vital Product Data)

ターゲットやロジカル・ユニットのベンダ固有情報を報告する機能として VPD が追加(オプション)されました。VPD には装置シリアル番号、センス・データで報告される FRU コードの説明の記述、CHANGE

DEFINITION コマンドで指定が可能な動作モードの記述などが定義されていますが、その他のベンダ固有の情報を追加することも可能です。インシエータは INQUIRY コマンドの CDB 上で、スタンダード INQUIRY データと VPD のどちらを要求するか指定します。

#### ● MODE SELECT/SENSE : 15/55/1A/5Ah

256 バイト長以上のパラメータ・リストを扱えるようにするため、従来の 6 バイト長 CDB に加えて 10 バイト長 CDB(グループ 2)が追加されました。グループ 2 の MODE SELECT/SENSE コマンドではパラメータのヘッダ部も 8 バイト長に拡張されています。

モード・パラメータの形式は SCSI1 ではベンダ固有になっていましたが、SCSI2 では CCS で規定されたページ形式が標準形になりました。また、新機能(統計情報、ECA、AEN、タグ付きキューイング)をコントロールするためのパラメータ・ページ(ページ A : Control Mode Page)などが追加になったほか、CCS と同一のパラメータ・ページでもページ長を拡張して機能を追加したものがあります。

モード・パラメータはターゲットによってサポート範囲が違ったり、将来の規格で機能が追加される可能性があります。インシエータは MODE SENSE コマンドを使用してターゲットがサポートしているパラメータの種類や長さとその属性を調べ、それに基づいて MODE SELECT コマンドを発行するようにし、特定の仕様レベルに依存しないようにしておくべきです。

#### ● SEND/RECEIVE DIAG. : 1D/1Ch

SEND DIAG.コマンドのセルフ・テスト機能のサポートは必須条件になっています。インシエータはこの

〈表 11〉  
全デバイス・タイプ共通の  
コマンド・セット

区分	OP コード	コマンド名称	SCSI2	CCS	SCSI1
グループ 0	00	TEST UNIT READY	M	M	O
	03	REQUEST SENSE	M	M	M
	12	INQUIRY	M	M	E
	15	MODE SELECT (6)	Z	O	O
	18	COPY	O	O	O
	1A	MODE SENSE (6)	Z	O	O
	1C	RECEIVE DIAGNOSTIC RESULTS	O	O	O
	1D	SEND DIAGNOSTIC	M	M	O
グループ 1	39	COMPARE	O	O	O
	3A	COPY AND VERIFY	O	O	O
	3B	WRITE BUFFER	O	O	RSVD
	3C	READ BUFFER	O	O	RSVD
グループ 2	40	CHANGE DEFINITION	O	RSVD	RSVD
	4C	LOG SELECT	O	RSVD	RSVD
	4D	LTG SENSE	O	RSVD	RSVD
	55	MODE SELECT (10)	Z	RSVD	RSVD
	5A	MODE SENSE (10)	Z	RSVD	RSVD

M : サポート必須, E : 拡張仕様, O : オプション,  
Z : デバイス・タイプ依存, RSVD : リザーブ



機能を使用してターゲットに自己診断を実行させることができます。自己診断の結果はステータスで通知されるのでセルフ・テスト機能だけを使用するときはRECEIVE DIAG. RESULTSは必要としません。

オプション機能としてSEND DIAG.コマンドでパラメータを送ってターゲットに特定の診断動作などを実行させ、その結果をRECEIVE DIAG. RESULTSコマンドで取り出すこともできます。このときのパラメータにもページ形式やターゲットのサポート範囲を調べる機能が導入され、装置個別仕様からの分離が図られています。

ダイレクト・アクセス・デバイスではこのオプション機能を使用してデータ・ブロックのアドレス変換(論理アドレス→物理アドレス)を行うための標準パラメータが規定されています。

● WRITE/READ BUFFER : 3B/3Ch

これらはCCSで追加されたコマンドですが、64Kバイト長以上のデータ・バッファを操作しやすいよう機能が拡張されました。

また、READ BUFFERコマンドでターゲットのデータ・バッファの構成を調べる機能が追加され、装置個別仕様からの独立性が高められています。さらに、WRITE BUFFERコマンドにはマイクロコードのダウン・ロード機能(イニシエータからターゲットのマイクロコードや制御情報を転送する機能)が追加され、オンラインでのバージョン・アップなどに対応できるようになっています。

● CHANGE DEFINITION : 40h

プログラマブル動作モードの項で述べたようにイニシエータはこのコマンドでターゲットとロジカル・ユニットの動作モードを、SCSI2, SCSI1, CCSのいずれか、あるいはベンダ固有の特殊動作モードに変更することを指定できます。

動作モードの変更適用範囲(指定されたロジカル・ユニットのみ/ターゲット全体、このコマンドを発行したイニシエータに対してのみ/すべてのイニシエータに対して)はターゲットの組み込み仕様に依存します。動作モードの変更が他のイニシエータからのコマンドの動作に影響を与えるときは、ターゲットは、それらのイニシエータに対してユニット・アテンション・コンディションを生成し、動作モードが変更されたことを通知します。

● LOG SELECT/SENSE : 4C/4Dh

これらのコマンドはターゲットが累積する統計情報を操作するものです。統計情報にはSCSIバスの動作状態に関連したバッファ・オーバラン/アンダランの発生回数や、I/O装置のメカニカル動作および媒体でのエラー発生回数などがあります。イニシエータはLOG SELECTコマンドで、個々の統計情報の初期化やしきい値を設定したり、累積の開始や停止を指示し、LOG SENSEコマンドで累積値を読み出すことができます。ターゲットは統計情報の対象となる事象の発生回数を累積し、しきい値または最大値に達するとユニット・アテンション・コンディションを生成することができます。

統計情報はSCSIバスの動作や負荷バランスをチュ

〈図 20〉  
センス・データの標準形式

ビット	7	6	5	4	3	2	1	0
バイト 0	Valid	エラーコード：70h または 71h						
バイト 1	セグメント番号(COPY, COMPARE, COPY AND VERIFY)							
バイト 2	Filemark	EOM	ILI	リザーブ=0	センス・キー			
バイト 3	[MSB]							
バイト 4	インフォメーション							
バイト 5								
バイト 6								
バイト 7	追加センス・データ長( $n-7$ )							
バイト 8	[MSB]							
バイト 9	コマンド固有インフォメーション (Command-Specific Information)							
バイト 10								
バイト 11								
バイト 12	ASC(Additional Sense Code)							
バイト 13	ASCQ(Additional Sense Code Qualifier)							
バイト 14	FRU コード(Field Replaceable Unit Code)							
バイト 15	SKSV	センス・キー固有インフォメーション						
バイト 16								
バイト 17								
バイト 18								
バイト $n$								

≈      ≈

}

追加センス・バイト  
 (コマンド固有、デバイス固有、またはベンダ固有)

≈

〈図 21〉スタンダード INQUIRY データ

ビット	7	6	5	4	3	2	1	0
バイト 0	Peripheral Qualifier		デバイス・タイプ・コード					
バイト 1	RMB		デバイス・タイプ修飾子					
バイト 2	ISO Version		ECMA Version			ANSI Version		
バイト 3	AENC	TrmIOP	(リザーブ : 00)		Response Data Format			
バイト 4	追加データ長 (n - 4)							
バイト 5	(リザーブ : 00h)							
バイト 6	(リザーブ : 00h)							
バイト 7	RelAdr	WBus32	WBus16	Syns	Linked	リザーブ:0	CmdQue	SftRe
8 15	ベンダ ID[ASCII]							
16 31	プロダクト ID[ASCII]							
32 36	プロダクト版数 [ASCII]							
36 55	ベンダ固有							
56 95	(リザーブ : 00h)							
96 n	ベンダ固有							

000 : 未適合  
001 : SCSI1  
010 : SCSI2

0000 : SCSI1  
0001 : CCS  
0010 : SCSI2

000 : 未適合  
001 : SCSI1  
010 : SCSI2

0000 : SCSI1  
0001 : CCS  
0010 : SCSI2

オプション機能のサポート・レベルを表示するビット (1 : サポート, 0 : 未サポート)  
AENC: ATN 受信機能 (プロセッサ・デバイスのみ)  
TrmIOP: TERMINATE I/O PROCESS メッセージ  
RelAdr : 相対アドレス機能 (CDBの "RelAdr" ビット)  
WBus32: 32ビット幅のWIDE転送  
WBus16: 16ビット幅のWIDE転送  
Sync: 同期モードのデータ転送  
Linked: コマンド・リンク機能 (CDBの "Link" ビット)  
CmdQue: タグ付きコマンド・キューイング  
SftRe: ソフト・リセット・コンディション

ーニングするための情報採取や I/O 装置の予防・保守  
のための情報採取などに効果的な機能です。

## ダイレクト・アクセス・デバイスの コマンド

ダイレクト・アクセス・デバイス用のコマンドの一覧  
を表 12 に示します。ダイレクト・アクセス・デバイス  
については CCS で標準のコマンド・セットが確立され  
ていたため、大きな変更はありませんが、

- ① キャッシュ制御機能の追加
- ② 診断系コマンドの強化
- ③ ディスク装置の回転同期 (スピンドル・シンク) 制御  
機能の追加

などが行われています。

### ● キャッシュ制御用のコマンド

キャッシュ・メモリを備えた装置の動作を制御する  
ために次の 3 つのコマンドが用意されました。

#### ▶ PRE-FETCH : 34h

このコマンドは CDB で指定された範囲のデータを

媒体からキャッシュ・メモリ上に読み込みます (イニシ  
エータには転送しない)。

ターゲットはキャッシュ上のロックされていない領  
域にデータを読み込み、指定されたすべてのデータが  
正常に読み込めたときには CONDITION MET ステ  
ータスを報告します。

#### ▶ SYNCHRONIZE CACHE : 35h

このコマンドは CDB で指定された範囲のデータ・  
ブロックについてキャッシュ・メモリ上の値と媒体上  
の値を一致させるものです。

ライト・バック・キャッシュでまだ媒体上に書き込ま  
れていないデータ・ブロック (このコマンドの CDB で  
の指定範囲内) があればターゲットはそのデータを  
媒体上に書き出します。

#### ▶ LOCK-UNLOCK CACHE : 36h

このコマンドはキャッシュ・メモリ内のデータに対  
してターゲットのキャッシュ・アルゴリズムによる追  
い出しを禁止 (ロック) または許可 (アンロック) します。

データの範囲は CDB 上の論理ブロック・アドレス  
とブロック数により指定しますが、このコマンドが発



行された時点でキャッシュ上に存在しているデータ・ブロックだけがロック/アンロックの対象になります。

ロックされたデータはこのコマンドでアンロックが指定されるまでキャッシュ上に保持されます。

#### ● キャッシュ制御用モード・セレクト・パラメータ

キャッシュ・メモリを備えた装置は、通常、READ コマンドで指定されたデータ・ブロックを媒体から読み出した後、**後続のデータ・ブロック(イニシエータからは要求されていない)**をキャッシュ上に先読みしますが、先読みの範囲や先読み動作とつぎのコマンド実行との優先度などを制御するためのモード・セレクト・パラメータ (ページ 8: Caching Page, 図 22) が追加されました。イニシエータはこのパラメータを使用して WRITE コマンドと READ コマンドのキャッシュ動作の禁止/許可を行うこともできます。

#### ● CDB 上のキャッシュ制御用フラグ

コマンドの発行単位でキャッシュの動作を制御できるようにするため以下のコマンドの CDB 上に 2 種類の**制御フラグ(DPO, FUA)**が追加されました。

- |                    |            |
|--------------------|------------|
| ・ READ (10)        | : DPO, FUA |
| ・ WRITE (10)       | : DPO, FUA |
| ・ WRITE AND VERIFY | : DPO      |
| ・ VERIFY           | : DPO      |

#### ▶ DPO(CDB バイト 1, ビット 4)

**DPO(Disable Page Out)**フラグはそのコマンドの実行によってすでにキャッシュ上にある他のデータを置き換えてよいかどうかを指定します。DPO が 1 のときにはそのコマンドでアクセスされたデータ・ブロックは最低位の優先順位でキャッシュに保持すればよいということを指定します。つまり、キャッシュ上に存在しているデータの追い出しは禁止されます。

そして DPO が 0 のときにはそのコマンドの実行時にキャッシュ・データの置き換えを行ってもよいことを示します。

#### ▶ FUA(CDB バイト 1, ビット 3)

**FUA(Force Unit Access)**フラグはそのコマンドを実行するときに媒体アクセスを強制するかどうかを指定します。

FUA が 0 のときには WRITE コマンドはキャッシュ上にデータを格納しただけで終了(ライト・バック・キャッシュ)し、READ コマンドは指定されたデータがキャッシュにヒットしていれば媒体をアクセスすることなくキャッシュからデータを転送します。

一方 FUA が 1 のときにはそのコマンドは必ず媒体をアクセスして実行することを指定します。WRITE コマンドは媒体上へのデータの書き込みが完了してから終了ステータスを報告します(ライト・スルー・キャッシュ)。READ コマンドはキャッシュにヒットしていても必ず媒体からデータを読み出します。

なお、VERIFY 系のコマンドには FUA フラグはありませんが、いつも FUA=1 であるものとして、つまり必ず媒体上のデータに対してベリファイ動作が行われます。ここで、**FUA=1 の READ コマンドまたは VERIFY 系のコマンドではコマンドの実行前に暗黙的に SYNCHRONIZE CACHE コマンドと同一の動作が実行されます。**つまり対象データがまだ媒体に書き込まれていない場合にはその書き込みを実行してからベリファイまたはリードを行います。

#### ● READ LONG/WRITE LONG : 3E/3Fh

これらのコマンドは媒体上のデータ・ブロックのデータ部とその**エラー訂正コード(ECC)**を直接リード/ライトします。

イニシエータはエラー・リカバリ処理のテストなどのために READ LONG コマンドで読み出したデータの一部を変更して WRITE LONG コマンドで書き戻すことにより、訂正可能または訂正不可能なエラーが発生するデータ・ブロックを媒体上に任意に作成することができます。

#### ● WRITE SAME : 41h

このコマンドはイニシエータから転送された 1 ブロック分のデータを媒体上の指定された領域に繰り返してライトします。

このとき、各データ・ブロックの先頭にそのブロックの論理アドレスまたは物理アドレスを書き込むこともできます。この機能はテストなどの目的で媒体を一定のデータ・パターンで初期化するときなどに便利です。

#### ● 回転同期(スピンドル・シンク)制御機能

ディスク・アレイなどのアプリケーションでは複数台のディスク装置のスピンドル・モータの回転を同期させることがあります。イニシエータからその制御を行うための機能としてモード・セレクト・パラメータのページ 4(Rigid Disk Drive Geometry Page)が拡張されました。

**モード・セレクト・コマンドによってスピンドル・シンクのモード(禁止、スレープ、マスタ、マスタ・コントロール)や回転同期のオフセット(基準信号への同期角度)を指定することができます。**また、回転同期の完了や同期異常はユニット・アテンション・コンディションで通知されます。

#### ● その他の拡張機能

▶ **モード・セレクト・パラメータ**: エラー・リカバリ用パラメータはリード/ライト用のパラメータ(ページ 1: Read-Write Error Recovery Page)とベリファイ用のパラメータ(ページ 7: Verify Error Recovery Page)に分離されました。

また、CDR [Constant Density Recording: ZBR (Zone Bit Recording)]とも呼ばれ、内周と外周シリンダで記憶密度(BPI)を均一化することで外周シリン

〈表 12〉 ダイレクト・アクセス・デバイスのコマンド・セット

\* : 全デバイス・タイプ共通のコマンド

区分	OP コード	コマンド名称	キャッシュ・フラグ	サポートの要否		
				SCSI2	CCS	SCSI1
グループ 0	00*	TEST UNIT READY		M	M	O
	01	REZERO UNIT		O	O	O
	02			V	V	V
	03*	REQUEST SENSE		M	M	M
	04	FORMAT UNIT		M	M	M
	05			V	V	V
	06			V	V	V
	07	REASSIGN BLOCKS		O	O	O
	08	READ (6)		M	M	M
	09			V	V	V
	0A	WRITE (6)		M	M	M
	0B	SEEK (6)		O	O	O
	0C~11			V	V	V
	12*	INQUIRY		M	M	E
	13			V	V	V
	14			V	V	V
	15*	MODE SELECT (6)		O	O	O
	16	RESERVE		M	M	O
	17	RELEASE		M	M	O
	18*	COPY		O	O	O
	19			V	V	V
グループ 1	1A*	MODE SENSE (6)		O	O	O
	1B	START/STOP UNIT		O	O	O
	1C*	RECEIVE DIAGNOSTIC RESULTS		O	O	O
	1D*	SEND DIAGNOSTIC		M	M	O
	1E	PREVENT-ALLOW MEDIUM REMOVAL		O	O	O
	1F			RSVD	RSVD	RSVD
	20~24			V	V	V
	25	READ CAPACITY		M	M	E
	26			V	V	V
	27			V	V	V
	28	READ (10)	DPO FUA	M	M	E
	29			V	V	V
	2A	WRITE (10)	DPO FUA	M	M	E
	2B	SEEK (10)		O	O	O
	2C			V	V	V
	2D			V	V	V
	2E	WRITE AND VERIFY	DPO	O	O	O
	2F	VERIFY	DPO	O	O	O
	30	SEARCH DATA HIGH		O	O	O
	31	SEARCH DATA EQUAL		O	O	O
	32	SEARCH DATA LOW		O	O	O
	33	SET LIMITS		O	O	O
	34	PRE-FETCH		O	RSVD	RSVD
	35	SYNCHRONIZE CACHE		O	RSVD	RSVD
	36	LOCK/UNLOCK CACHE		O	RSVD	RSVD
	37	READ DEFECT DATA		O	O	RSVD
	38			RSVD	RSVD	RSVD
	39*	COMPARE		O	O	O
	3A*	COPY AND VERIFY		O	O	O
	3B*	WRITE BUFFER		O	O	RSVD
	3C*	READ BUFFER		O	O	RSVD
	3D			RSVD	RSVD	RSVD
グループ 2	3E	READ LONG		O	RSVD	RSVD
	3F	WRITE LONG		O	RSVD	RSVD
	40*	CHANGE DEFINITION		O	RSVD	RSVD
	41	WRITE SAME		O	RSVD	RSVD
	42~4B			RSVD	RSVD	RSVD
	4C*	LOG SELECT		O	RSVD	RSVD
	4D*	LOG SENSE		O	RSVD	RSVD
	4E~54			RSVD	RSVD	RSVD
	55*	MODE SELECT (10)		O	RSVD	RSVD
	56~59			RSVD	RSVD	RSVD
グループ 3, 4, 5	5A*	MODE SENSE (10)		O	RSVD	RSVD
	5B~5F			RSVD	RSVD	RSVD
	60~BF			RSVD	RSVD	RSVD
グループ 6, 7	CO~FF			V	V	V

M : サポート必須, E : 拡張仕様, O : オプション, RSVD : リザーブ  
V : ベンダ固有



〈図 22〉 キャッシュ制御用のモード・セレクト・パラメータ

ビット	7	6	5	4	3	2	1	0
バイト 0	PS	リザーブ: 0	ページ・コード: 08h					
バイト 1	ページ長: 0Ah							
バイト 2	(リザーブ: 00000)					WCE	MF	RCD
バイト 3	リード・データの保持優先度 (Demand Read Retention Priority)				ライト・データの保持優先度 (Write Retention Priority)			
バイト 4	プリフェッチ抑止ブロック数 (Disable Pre-fetch Transfer Length)							
バイト 5								
バイト 6	最小プリフェッチ (Minimum Pre-fetch)							
バイト 7								
バイト 8	最大プリフェッチ (Maximum Pre-fetch)							
バイト 9								
バイト 10	最大プリフェッチ制限ブロック数 (Maximum Pre-fetch Ceiling)							
バイト 11								

- WCE  
0: ライト・スルー  
1: ライト・バック
- RCD  
0: キャッシュ許可  
1: キャッシュ禁止

- リード・データの保持優先度  
0: 他のデータと同列  
1: 低優先度  
F: 高優先度
- ライト・データの保持優先度  
0: 他のデータと同列  
1: 低優先度  
F: 高優先度
- READ コマンド終了後のキャッシュへの先読み動作
  - ・先読みの許可: (RCD=0) & (READ コマンドのブロック数 ≤ プリフェッチ抑止ブロック数)
  - ・最小プリフェッチ量:
    - ・ If MF=0, = (最小プリフェッチ)
    - ・ If MF=1, = (最小プリフェッチ) \* (READ コマンドのブロック数)
  - ・最小プリフェッチ量分の先読み動作は、次のコマンドより優先
  - ・最大プリフェッチ量:
    - ・ If MF=0, = (最大プリフェッチ)
    - ・ If MF=1, = (最大プリフェッチ) \* (READ コマンドのブロック数)
  - ・ただし、上限は(最大プリフェッチ制限ブロック数)

キャッシュ機能をさらに高度化するため ANSI ではこのページの機能拡張  
(セグメント・キャッシュなど)を検討している

ダのトラック容量を増加させ、大容量化を図る手法)に関連したパラメータの追加や、フレキシブル・ディスク用のパラメータの機能追加も行われています。**CCS** で規定されたパラメータ・ページの内、ページ 1, 2, 4, 5 などは機能追加によりページ長が変更されているので注意が必要です。

▶ **FORMAT UNIT**: フォーマットिंगのときに媒体に書き込む“初期化データ・パターン”を指定する機能やイミディエート機能(CDB とパラメータの転送が終わった時点で終了ステータスを報告する)などが追加されました。

▶ **RESERVE/RELEASE**: 異なったイニシエータからのリザーブ要求をキューイングする機能は削除されました。また、サード・パーティ・リザーブのときに、RESERVE コマンドを発行したイニシエータのモード・パラメータをサード・パーティ・デバイス用のモード・パラメータにコピーする機能が設けられました。

\* \* \*

1990 年の中頃から **FAST SCSI** や **キュー・タグ・メッセージを組み込んだプロトコル IC** が登場しはじめており、最近では **WIDE SCSI** への対応も図られてきているので本格的な **SCSI2** 仕様の装置が続々と登

場してくることは間違いありません。

しかし、タグ付きコマンド・キューイングなどの新しい機能はソフトウェアの対応も含めて徐々にシステムへの組み込みが進んでいくと思われるので、実使用環境に適した標準形が確立され一般化するまでにはもう少し時間がかかると予想されます。

ANSI ではすでに、SCSI2 の機能をさらに発展させた **SCSI3** の検討が開始されています。今後ますます進展が予想されるなかで SCSI2 への取り組みは必要不可欠になっています。

●参考・引用\*文献●

- (1)\*ANSI, X3T9.2/86-109 Revision 10c, X3T9/89-042, draft proposed American National Standard for Information Systems - Small Computer System Interface-2(SCSI-2), American National Standard Committee X3, March 9, 1990.
- (2) ANSI, X3.131-1986, American National Standard for Information Systems - Small Computer System Interface (SCSI), American National Standard Institute, 1986.
- (3) ANSI, X3T9.2/85-52, Common Command Set(CCS) of the Small Computer System Interface (SCSI), REV 4.B. American National Standard Committee X3, 1986.
- (4)\*菅谷誠一;「SCSI 規格の詳細解説」, 最新 SCSI マニュアル, CQ 出版株式会社.

## 編集雑記

### 編集部から

● ハードディスクは本文中でも解説されているように非常に高度な技術を使っています。

そのようなわけで実際、ハードディスク・ユニットそのものを作ることやユニットの中身を改造することなどは個人の作業がおよぶ領域ではありません。もはや入出力がはっきりとわかっているブラックボックスとしてあつかうしかありません。

それと同じく製品としてのハードディスクの見た目もただの箱です。しかも外付け型ならば箱として見えるものの内蔵型ともなるとあるのかないのか全く判別が付きません。

このようにハードディスクにはある種の得体の知れなさがあります。少なくともこの文章を書いている本人はそう思っています。

こういった得体の知れなさがアマチュア製作への取り付きづらさや、そもそもハードディスクを個人でいじくり回してみようという気にさせてくれない原因になっているのではないかと考えます。

しかし、ハードディスク・ユニットの中身に手を加えることが不可能だとしても、取り付くしまが全くな

いわけではないです。ユニットの外側にならいくらでも手を加える余地が残っています。

● そのような理由で今回の特集ではハードディスク自身についての解説は第一章から第三章で一通り行なうだけにしました。他の章はすべてハードディスクをつなぐインターフェースである SCSI の解説と使い方は。

第五章および第六章は現在個人で扱えるハードディスク関連の題材としてはかなりいい線を行っているような気がします。

もっともハードウェアの最近の傾向である 1 チップ化の波はここでも例外ではなく、ハードウェアは機能 IC のつなぎ合わせで残りはすべてソフトウェアという手法を取るのには仕方のないことです。

● 今回のページの都合上、具体的な製作例を紹介することができませんでした。そういった事例に関しては本誌「トランジスタ技術」誌上でおいおい御紹介できると思います。

また、第六章として掲載した内容はもともと NIFTY-SERVE の FDEVICE という電子会議をはじめとしたパソコンネット上で企画、検討、製作されたものであることをこ

こでおことわりしておきます。興味を持たれた方はこちらをのぞいてみるのもいかがでしょうか。第六章で紹介したソフト&ハードはもちろんのこと、その他にも興味深い内容が数多くアップされています。

● SCSI2 によって WIDE と FAST の二種類の拡張がなされた SCSI は、これまで以上に使用範囲が広がるはずですが、ハードディスクの他、考えられるだけでも MO, MT, DAT, CD-ROM, WORM, プリンタ, プロッタ, イメージ・スキャナ, 他のマイコン・システムなどさまざまです。マイコンの大容量化, 高速化がなされていく以上, 高速インターフェースの需要が無くなるわけがありません。

今、例に挙げたもののうち、いくつかはまだ製品として市場に出回っていません。それらは次の主流としてひかえているものばかりです。いずれ、それらが市場に出回るようになる頃、SCSI は現在のセントロニクス・バスや RS 232 C のように誰もが簡単に製作できる入門に最適な簡単明瞭インターフェースと言われるようになるのでしょうか？ これが編集を終えた後、私がいだいた、さやかな疑問です。 (▲)

次号のお知らせ(6月28日発売)

### 特集 新・電源回路設計のすべて

あらゆる電子装置に必要な電源回路技術を、実際に設計・製作するプロセスを通じて詳解します。

3端子レギュレータ回路や低損失型のリニア電源、RCC方式やフォワード・コンバータ方式などのSW電源、また近年注目されている共振型SW電源について詳しく説明します。また、電源用部品や新しい安全規格の解説も行う予定です。

## トランジスタ技術 SPECIAL No. 27

発行所 CQ出版株式会社 Printed in Japan

〒170 東京都豊島区巣鴨 1-14-2

電話 編集部: 03(5395)2123, 広告部: 03(5395)2132

営業部: 03(5395)2141

振替 東京 0-10665

発行人 神戸一夫

編集人 蒲生良治

© CQ出版株式会社 1991(定価は表四に表示してあります)

1991年5月1日発行

印刷・製本 三晃印刷株式会社



## 解析 ノイズ・メカニズム

### ●雑音発生の原因追求と誤動作防止対策

ノイズ対策技術を、理論的にも明解にかつわかりやすく紹介したのが本書の特徴です。

岡村 迪夫 著  
A 5 判 356頁  
定価1,960円  
送料310円

## ASICの論理回路設計法

### ●スーパーマシンのためのデジタル・システム設計ノウハウ

デジタル・システム(ASIC)の設計に欠かせない古典的回路技術を新感覚で集大成しました。

小林 芳直 著  
A 5 判 288頁  
定価1,960円  
送料260円

## マイコン・システムのリアルタイム制御作法

### ●Z80の組み込みコンピュータ設計ノウハウ

本書はZ80 CPUにターゲットを置いて、装置内に組み込まれ利用されているマイコン・システムの割り込み制御からリアルタイム処理技術までを紹介します。

宮崎 誠一 共著  
宮崎 仁  
A 5 判 248頁  
定価1,648円  
送料260円

## 新・低周波/高周波回路設計マニュアル

### ●増幅回路の設計法から実装ノウハウまで

本書は、ICやトランジスタやFETを使った低周波増幅回路の設計法から、高周波トランジスタの使い方、高周波変復調回路の実例などを採り上げています。

鈴木 雅臣 著  
A 5 判 288頁  
定価1,960円  
送料260円

## PLDの論理回路設計法

### ●ASIC時代に備えるデジタル回路設計ノウハウ

デジタル回路のコンパクト化に欠かせないPLDの実戦応用のための待望の書籍です。

小林 芳直 著  
A 5 判 272頁  
定価1,850円  
送料260円

## 実用アナログ・フィルタ設計法

### ●信号処理を正しく実現するために

本書では、第1部でアクティブ・フィルタの設計法を、第2部では、LC形を中心に、より高度な回路変換方法などを詳しく解説しました。

今田 悟 共著  
深谷 武彦  
A 5 判 328頁  
定価2,580円  
送料260円

## 物理計測システム実用設計

### ●電子回路とパソコンによる計測技術ノウハウ

本書は、各種実験装置やラボオート・システムを初心者でも簡単に作れるように解説しました。

物理教材研究会 編  
A 5 判 280頁  
定価2,000円  
送料260円

## アナログ回路のグレードアップ技法

### ●いかにしてローコストで高性能化を図るか

本書は、アナログ回路をあらゆる角度から再検討し、いかにしてローコストで高性能を得るかについて、実験の裏付けをもとにわかりやすく解説をしています。

中野 正次 著  
A 5 判 308頁  
定価2,400円  
送料260円

